DATA STRUCTURES

LAB MANUAL

Subject Code:	A30582
Regulations:	R13 – JNTUH
Class:	II Year I Semester (CSE)

Prepared By

Ms. B Padmaja
Associate Professor, CSE

Mr. CH Suresh Kumar Raju
Associate Professor, CSE

Mr. Hare Krishna Allu
Assistant Professor, CSE

Department of Computer Science & Engineering

INSTITUTE OF AERONAUTICAL ENGINEERING

Dundigal – 500 043, Hyderabad

INSTITUTE OF AERONAUTICAL ENGINEERING

Dundigal, Hyderabad - 500 043

COMPUTER SCIENCE AND ENGINEERING

Program Outcomes	
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
Program Specific Outcomes	
PSO1	**Professional Skills:** The ability to research, understand and implement computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient analysis and design of computer-based systems of varying complexity.
PSO2	**Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.
PSO3	**Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths, to be an entrepreneur, and a zest for higher studies.

DATA STRUCTURES LAB SYLLABUS

Recommended Systems/Software Requirements:
Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100MB free disk space. C compiler.

S. No.	List of Experiments	Page No.
1	Write a C program that uses functions to perform the following: a) Create a singly linked list of integers. b) Delete a given integer from the above linked list. c) Display the contents of the above list after deletion.	1
2	Write a C program that uses functions to perform the following: a) Create a doubly linked list of integers. b) Delete a given integer from the above doubly linked list. c) Display the contents of the above list after deletion.	6
3	Write a C program that uses stack operations to convert a given infix expression into its postfix Equivalent, Implement the stack using an array.	11
4	Write C programs to implement a double ended queue ADT using i) array and ii) doubly linked list respectively.	14
5	Write a C program that uses functions to perform the following: a) Create a binary search tree of characters. b) Traverse the above Binary search tree recursively in Postorder.	23
6	Write a C program that uses functions to perform the following: a) Create a binary search tree of integers. b) Traverse the above Binary search tree non recursively in inorder.	26
7	Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order: a) Insertion sort b) Merge sort	29
8	Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order: a) Quick sort b) Selection sort	32
9	i) write a C program to perform the following operation: A) Insertion into a B-tree ii) Write a C program for implementing Heap sort algorithm for sorting a given list of integers in ascending order.	35
10	Write a C program to implement all the functions of a dictionary (ADT) using hashing.	41
11	Write a C program for implementing Knuth-Morris- Pratt pattern matching algorithm.	46
12	Write C programs for implementing the following graph traversal algorithms: a)Depth first traversal b)Breadth first traversal	49
Content Beyond Syllabi		
1	*Write a C Program to check whether two given lists are containing the same data.	52
2	*Write a C program to find the largest element in a given doubly linked list.	54
3	*Write a C program to reverse the elements in the stack using recursion.	56
4	*Write a C program to implement stack using linked list.	58
5	*Write a C program to count the number of nodes in the binary search tree.	62
6	*Write a C program to sort an array of integers in ascending order using radix sort.	64
7	*Write a C program to sort a given list of strings.	66

*****Content beyond the university prescribed syllabi

ATTAINMENT OF PROGRAM OUTCOMES & PROGRAM SPECIFIC OUTCOMES

Exp. No.	Experiment	Program Outcomes Attained	Program Specific Outcomes Attained
1	Write a C program that uses functions to perform the following: a) Create a singly linked list of integers. b) Delete a given integer from the above linked list. c) Display the contents of the above list after deletion.	PO1, PO2	PSO1
2	Write a C program that uses functions to perform the following: a) Create a doubly linked list of integers. b) Delete a given integer from the above doubly linked list. c) Display the contents of the above list after deletion.	PO1, PO2	PSO1
3	Write a C program that uses stack operations to convert a given infix expression into its postfix equivalent. Implement the stack using an array.	PO1, PO2	PSO1, PSO2
4	Write C programs to implement a double ended queue ADT using i) array and ii) doubly linked list respectively.	PO1, PO2	PSO1
5	Write a C program that uses functions to perform the following: a) Create a binary search tree of characters. b) Traverse the above Binary search tree recursively in Postorder.	PO1, PO2, PO3	PSO1, PSO2
6	Write a C program that uses functions to perform the following: a) Create a binary search tree of integers. b) Traverse the above Binary search tree non recursively in inorder.	PO1, PO2, PO3	PSO1
7	Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order: b) Insertion sort b) Merge sort	PO1, PO2	PSO1
8	Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order: b) Quick sort b) Selection sort	PO1, PO2	PSO1
9	i) write a C program to perform the following operation: A) Insertion into a B-tree ii) Write a C program for implementing Heap sort algorithm for sorting a given list of integers in ascending order.	PO1, PO2, PO3	PSO1
10	Write a C program to implement all the functions of a dictionary (ADT) using hashing.	PO1, PO2, PO3	PSO1, PSO2
11	Write a C program for implementing Knuth-Morris- Pratt pattern matching algorithm.	PO1, PO2, PO4, PO12	PSO1, PSO2
12	Write C programs for implementing the following graph traversal algorithms: a)Depth first traversal b)Breadth first traversal	PO1, PO2, PO3	PSO1
Content Beyond Syllabi			
1	*Write a C Program to check whether two given lists are containing the same data.	PO1, PO2	PSO1
2	*Write a C program to find the largest element in a given doubly linked list.	PO1, PO2	PSO1
3	*Write a C program to reverse the elements in the stack using recursion.	PO1, PO2	PSO1, PSO2

Exp. No.	Experiment	Program Outcomes Attained	Program Specific Outcomes Attained
4	*Write a C program to implement stack using linked list.	PO1	PSO1
5	*Write a C program to count the number of nodes in the binary search tree.	PO1, PO2	PSO1
6	*Write a C program to sort an array of integers in ascending order using radix sort.	PO1, PO2	PSO1
7	*Write a C program to sort a given list of strings.	PO1	PSO1

*****Content beyond the University prescribed syllabi

DATA STRUCTURES LABORATORY

OBJECTIVE:

The objective of this lab is to teach students various data structures and to explain them algorithms for performing various operations on these data structures. This lab complements the data structures course. Students will gain practical knowledge by writing and executing programs in C using various data structures such as arrays, linked lists, stacks, queues, trees, graphs, hash tables and search trees.

OUTCOMES:

Upon the completion of Data Structures practical course, the student will be able to:

1. **Design** and analyze the time and space efficiency of the data structure.
2. **Identity** the appropriate data structure for given problem.
3. **Understand** the applications of data structures.
4. **Choose** the appropriate data structure and algorithm design method for a specified application.
5. **Understand** which algorithm or data structure to use in different scenarios.
6. **Understand** and apply fundamental algorithmic problems including Tree traversals, Graph traversals.
7. **Compare** different implementations of data structures and to recognize the advantages and disadvantages of them.
8. **Write** complex applications using structured programming methods.

EXPERIMENT 1

1.1 OBJECTIVE

1. To create a singly linked list of integers.
2. Delete a given integer from the above linked list.
3. Display the contents of the above list after deletion.

1.2 RESOURCE:

Turbo C

1.3 PROGRAM LOGIC

1. Create a node using structure
2. Dynamically allocate memory to node
3. Create and add nodes to linked list

1.4 PROCEDURE

Go to debug -> run or press CTRL + F9 to run the program.

1.5 SOURCE CODE

program to create a single linked list, delete the contents and display the contents

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
/*declaring a structure to create a node*/
struct node
{
    int data;
    struct node *next;
};
struct node *start;

/* inserting nodes into the list*/
/*function to insert values from beginning of the the single linked
list*/ void insertbeg(void)
{
    struct node *nn;
    int a;
    /*allocating implicit memory to the node*/
    nn=(struct node *)malloc(sizeof(struct node));
    printf("enter data:");
    scanf("%d",&nn->data);
    a=nn->data;
    if(start==NULL) {        /*checking if List is empty*/

        nn->next=NULL;
        start=nn;
    }
    else
    {
        nn->next=start;
        start=nn;
    }
    printf("%d succ. inserted\n",a);
    return;
}
/*function to insert values from the end of the linked list*/
```

```
void insertend(void)
{
    struct node *nn,*lp;int b;
    nn=(struct node *)malloc(sizeof(struct node));
    printf("enter data:");
    scanf("%d",&nn->data);
    b=nn->data;
    if(start==NULL)
    {
        nn->next=NULL;
        start=nn;
    }
    else
    {
        lp=start;
        while(lp->next!=NULL)
        {
            lp=lp->next;
        }
        lp->next=nn;
        nn->next=NULL;
    }
    printf("%d is succ. inserted\n",b);
    return;
}
```

/*function to insert values from the middle of the linked list*/ void insertmid(void)

```
{
    struct node *nn,*temp,*ptemp;int x,v;
    nn=(struct node *)malloc(sizeof(struct node));
    if(start==NULL)
    {
        printf("sll is empty\n"); return;
    }
    printf("enter data before which no. is to be inserted:\n");
    scanf("%d",&x);
    if(x==start->data)
    {
        insertbeg();
        return;
    }
    ptemp=start;
    temp=start->next;
    while(temp!=NULL&&temp->data!=x)
    {
        ptemp=temp;
        temp=temp->next;
    }
    if(temp==NULL)
    {
        printf("%d data does not exist\n",x);
    }
    else
    {
            printf("enter data:");
            scanf("%d",&nn->data);
            v=nn->data;
            ptemp->next=nn;
```

```
            nn->next=temp;
            printf("%d succ. inserted\n",v);
    }
    return;
}
/*deletion operation*/
void deletion(void)
{
    struct node *pt,*t;
    int x;
    if(start==NULL)
    {
        printf("sll is empty\n");
       return;
    }
    printf("enter data to be deleted:");
    scanf("%d",&x);
    if(x==start->data)
    {
        t=start;
        /* assigning first node pointer to next nod pointer to delete a
       data from the starting of the node*/
        start=start->next;
        free(t);
        printf("%d is succ. deleted\n",x);
        return;
    }
    pt=start;
    t=start->next;
    while(t!=NULL&&t->data!=x)
    {
        pt=t;t=t->next;
    }
    if(t==NULL)
    {
        printf("%d does not exist\n",x);return;
     }
    else
    {
        pt->next=t->next;
    }
    printf("%d is succ. deleted\n",x);
    free(t);
    return;
}
void display(void)
{
    struct node *temp;
    if(start==NULL)
    {
        printf("sll is empty\n");
        return;
    }
    printf("elements are:\n");
    temp=start;
    while(temp!=NULL)
    {
        printf("%d\n",temp->data);
```

```
            temp=temp->next;
        }
        return;
    }
    /* main program*/
    int main()
    {
        int c,a;    start=NULL;
        do
        {
            printf("1:insert\n2:delete\n3:display\n4:exit\nenter choice:");
            scanf("%d",&c);
            switch(c)
            {
                case 1:
                printf("1:insertbeg\n2:insert end\n3:insert mid\nenter choice:");
                scanf("%d",&a);
                switch(a)
                {
                     case 1:insertbeg(); break;
                    case 2:insertend(); break;
                    case 3:insertmid(); break;
                }
                break;
                case 2:deletion(); break;
                case 3:display(); break;
                case 4:printf("program ends\n");break;
                default:printf("wrong choice\n");
                break;
            }
        }while(c!=4);return 0;
    }
```

1.6 PRE LAB QUESTIONS

1. What is data structure
2. How the memory is allocated dynamically
3. What is linked list
4. What is node
5. What are the types of linked list

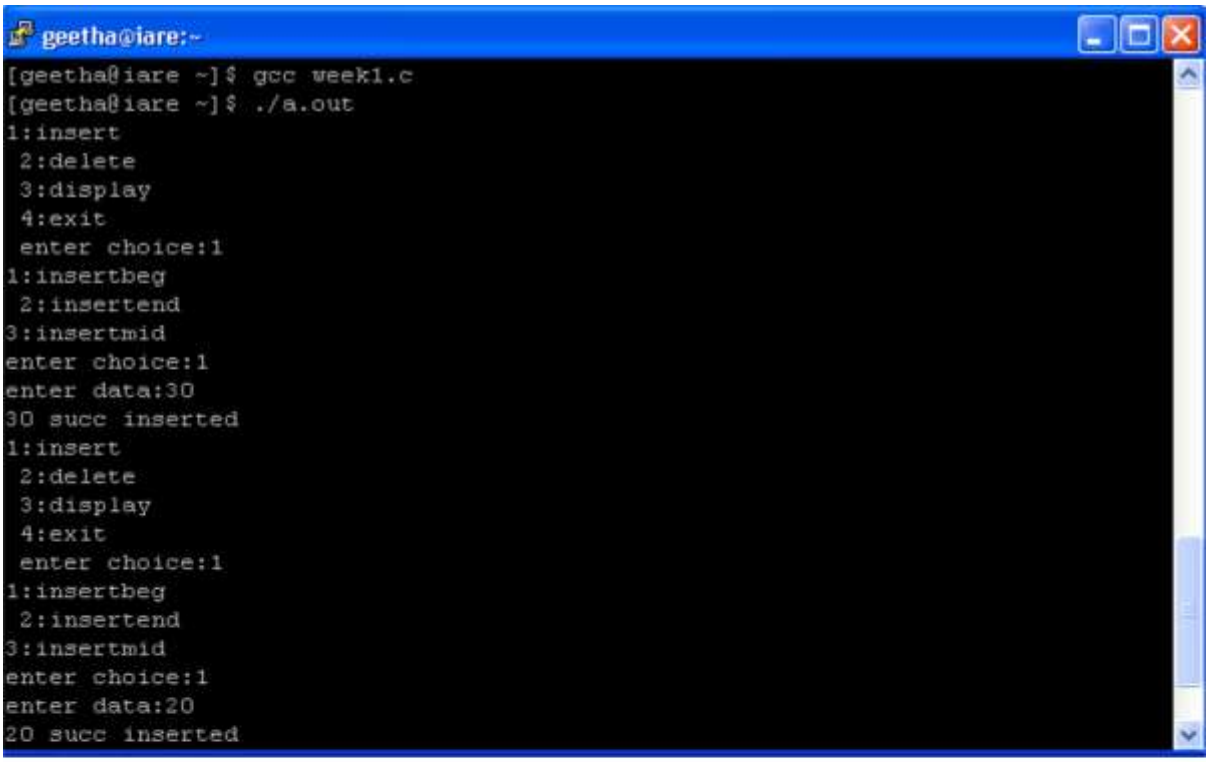
1.7 LAB ASSIGNMENT

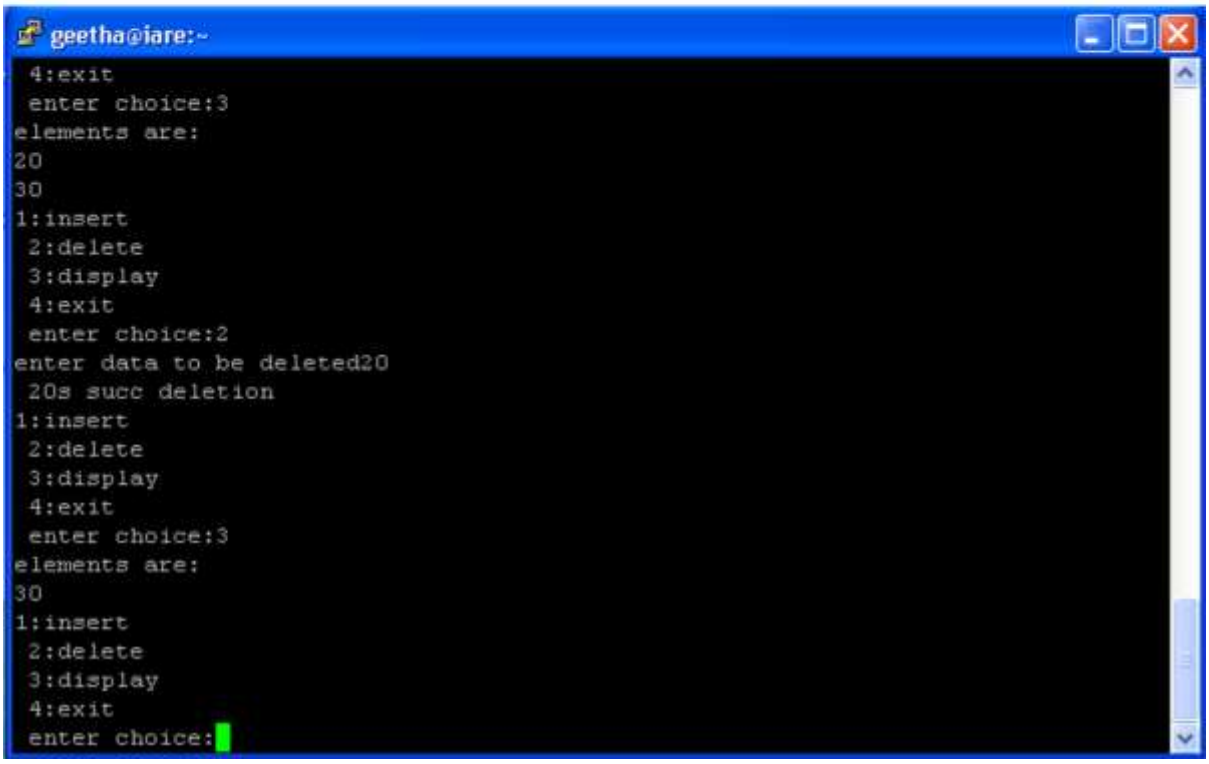
1. Write a program to insert a node at first , last and at specified position of linked list
2. Write a program to delete a node from first, last and at specified position of linked list

1.8 POST LAB QUESTIONS

1. How to represent linked list
2. How will you traverse linked list in reverse order
3. List the advantages and disadvantages of linked list?

1.9 INPUT AND OUTPUT





EXPERIMENT 2

2.1 OBJECTIVE

1. Create a doubly linked list of integers.
2. Delete a given integer from the above doubly linked list.
3. Display the contents of the above list after deletion.

2.2 RESOURCE

Turbo C

2.3 PROGRAM LOGIC

1. Create a node using structure
2. Dynamically allocate memory to node
3. Create and add nodes to linked list

2.4 PROCEDURE

Go to debug -> run or press CTRL + F9 to run the program

2.5 SOURCE CODE

Program to create a double linked list to inserting, deleting and displaying the contents

```
#include<stdio.h>
#include<stdlib.h>
/*declaring a structure to create a node*/
struct node
{
     struct node *prev;
     int data;
     struct node *next;
};
struct node *start,*nt;
/* inserting nodes into the list*/
/*function to insert values from beginning of the the double linked list*/

void insertbeg(void)
{
    int a;
    struct node *nn,*temp;
/*allocating implicit memory to the node*/
    nn=(struct node *)malloc(sizeof(struct node));
    printf("enter data:");
    scanf("%d",&nn->data);
    a=nn->data;
  if(start==NULL) {   /*checking if List is empty*/
      nn->prev=nn->next=NULL;
      start=nn;
  }
  else
  {
     nn->next=start;
     nn->prev=NULL;
     start->prev=nn;
     start=nn;
  }
  printf("%d succ inserted \n",a);
```

```
}
/*function to insert values from the end of the linked list*/

void insertend(void)
{
    int b;
    struct node *nn,*lp;
    nn=(struct node *)malloc(sizeof(struct node));
    printf("enter data:");
    scanf("%d",&nn->data);
    b=nn->data;
    if(start==NULL)
  {
/* assigning first node pointer to next nod pointer to delete a data from the
   starting of the node*/

       nn->prev=nn->next=NULL;
       start=nn;
  }
  else
  {
     lp=start;
    while(lp->next!=NULL)
   {
      lp=lp->next;
    }
    nn->prev=lp;
    lp->next=nn;
    nn->next=NULL;
  }
 printf("%d succ inserted\n",b);
}
/*function to insert values from the middle of the linked list*/

void insertmid(void)
{
    struct node *nn,*temp,*ptemp;
    int x,c;
   if(start==NULL)
   {
       printf("dll is empty\n");
      return;
   }
    printf("enter data before which nn is to be inserted\n");
    scanf("%d",&x);
    if(x==start->data)
    {
       insertbeg();
     }
     ptemp=start;
     temp=start->next;
     while(temp->next!=NULL&&temp->data!=x)
    {
      ptemp=temp;
      temp=temp->next;
    }
    if(temp==NULL)
   {
```

```
        printf("%d does not exit\n",x);
      }
      else
     {
/*allocating implicit memory to the node*/

         nn=(struct node *)malloc(sizeof(struct node));
         printf("enter data");
         scanf("%d",&nn->data);
        c=nn->data;
        nn->data;
       nn->prev=ptemp;
       nn->next=temp;
       ptemp->next=nn;
       temp->prev=nn;
       printf("%d succ inserted \n",c);
   }
}
/*end of insertion operation*/
/*deletion operation*/
void deletion()
{
     struct node *pt,*t;
     int x;
     t=pt=start;
     if(start==NULL)
    {
        printf("dll is empty\n");
     }
     printf("enter data to be deleted:");
    scanf("%d",&x);
    if(x==start->data)
   {
      t=start;
      t=t->next;
      free(start);
      start=t;
      start=pt;
   }
   else
  {
     while(t->next!=NULL&&t->data!=x)
    {
       pt=t;    /*logic for traversing*/
       t=t->next;
    }
   if(t->next==NULL&&t->data==x)
   {
      free(t);
      pt->next=NULL;
   }
   else
   {
      if(t->next==NULL&&t->data!=x)
          printf("data not found");
     else
      {
        pt->next=t->next;
```

```
        free(t);
      }
   }
   printf("%d is succ deleted\n",x);
  }
}
/*end of deletion operation*/
/*display operation*/
void display()
{
    struct node *temp;
    if(start==NULL)
    printf("stack is empty ");
    temp=start;
    while(temp->next!=NULL)
   {
      printf("%d",temp->data);
      temp=temp->next;
   }
   printf("%d",temp->data);
 }
/*end of display operation*/
/*main program*/
int main()
{
    int c,a;
    start=NULL;
   do
  {
      printf("1.insert\n2.delete\n3.display\n4.exit\nenter choice:");
      scanf("%d",&c);
      switch(c)
     {
        case 1:printf("1.insertbeg\n2.insertend\n3.insertmid\nenter choice:");
                scanf("%d",&a);
                switch(a)
               {
                     case 1:insertbeg();
                            break;
                     case 2:insertend();
                            break;
                     case 3:insertmid();
                            break;
               }
              break;
         case 2:deletion();
                break;
         case 3:display();
                break;
      case 4:printf("program ends\n");
                break;
      default:printf("wrong choice\n");
                break;
     }
  }
  while(c!=4);
  return 0;
 }
```

2.6 PRE LAB QUESTIONS

1. What is double linked list
2. How to represent a node in double linked list
3. Differentiate between single and double linked list

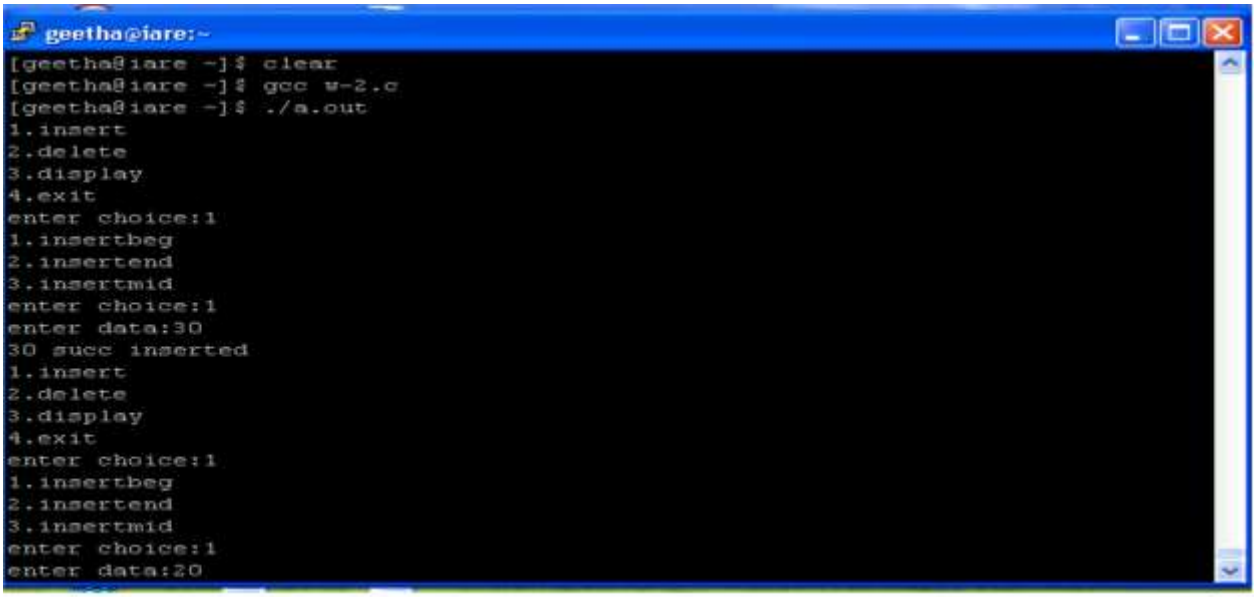
2.7 LAB ASSIGNMENT

1. Write a program to insert a node at first , last and at specified position of double linked list
2. Write a program to eliminate duplicates from double linked list
3. Write a program to delete a node from first, last and at specified position of double linked list

2.8 POST LAB QUESTIONS

1. How to represent double linked list
2. How will you traverse double linked list
3. List the advantages of double linked list over single list

2.9 INPUT AND OUTPUT



EXPERIMENT 3

3.1 OBJECTIVE
To convert a given infix expression into its postfix Equivalent, Implement the stack using an array**.**

3.2 RESOURCE:
Turbo C

3.3 PROGRAM LOGIC
1. Create a stack
2. Read an infix expression
3. convert infix exression into postfix expression

3.4 PROCEDURE:
Go to debug -> run or press CTRL + F9 to run the program

3.5 SOURCE CODE:

Program to convert a given infix expression into its postfix Equivalent, Implement the stack using an array.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define MAX 20
char stack[MAX];
int top=1;
char pop(); /*declaration of pop function*/
void push(char item); /*declaration of push function*/
int prcd(char symbol) /*checking the precedence*/
{
  switch(symbol) /*assigning values for symbols*/
  {
     case '+':
     case '-': return 2;
     break;
     case '*':
     case '/': return 4;
     break;
     case '^':return 6;
     break;
     case '(':
     case ')':
     case '#':return 1;
    break;
  }
}
int(isoperator(char symbol))  /*assigning operators*/
{
  switch(symbol)
  {
      case '+':
      case '*':
```

```
      case '-':
      case '/':
      case '^':
      case '(':
      case ')':return 1;
      break;
      default:return 0;
   }
 }
/*converting infix to postfix*/
void convertip(char infix[],char postfix[])
{
 int i,symbol,j=0;
 stack[++top]='#';
 for(i=0;i<strlen(infix);i++)
 {
  symbol=infix[i];
  if(isoperator(symbol)==0)
  {
     postfix[j]=symbol;
      j++;
  }
  else
  {
     if(symbol=='(')
     push(symbol);  /*function call for pushing elements into the stack*/
     else if(symbol==')')
     {
       while(stack[top]!='(')
      {
         postfix[j]=pop();
         j++;
      }
      pop();     /*function call for popping elements into the stack*/
    }
   else
   {
       if(prcd(symbol)>prcd(stack[top]))
       push(symbol);
       else
       {
         while(prcd(symbol)<=prcd(stack[top]))
         {
           postfix[j]=pop();
            j++;
         }
        push(symbol);
       }/*end of else loop*/
    }/*end of else loop*/
    } /*end of else loop*/
  }/*end of for loop*/
  While (stack[top]!='#')
  {
    postfix[j]=pop();
    j++;
  }
  postfix[j]='\0'; /*null terminate string*/
 }
```

```
/*main program*/
void main()
 {
     char infix[20],postfix[20];
     printf("enter the valid infix string \n");
     gets(infix);
     convertip(infix,postfix); /*function call for converting infix to postfix */
    printf("the corresponding postfix string is:\n"); puts(postfix);

  }
/*push operation*/
void push(char item)
{
   top++;
   stack[top]=item;
 }
/*pop operation*/
char pop()
{
   char a;
   a=stack[top];
   top--;
   return a;
 }
```

3.6 PRE LAB QUESTIONS

1. what is an expression
2. what are infix, prefix and postfix notations
3. what are polish and reverse polish notations
4. which data structure is used for infix to postfix conversion

3.7 LAB ASSIGNMENT

1. Convert the infix expression (a+b)-(c*d) into post fix form?
2. Convert the following expression A + (B * C) - ((D * E + F) / G) into post form.

3.8 POST LAB QUESTIONS

1. how to represent stack
2. why reverse polish notation is required
3. can we evaluate polish notation

3.9 INPUT AND OUTPUT

EXPERIMENT 4

4.1 OBJECTIVE

1. To implement a double ended queue ADT using arrays.
2. To implement a double ended queue ADT using doubly linked list.

4.2 RESOURCE:

Turbo C

4.3 PROGRAM LOGIC

Double ended queue ADT using arrays

1. Create a linked list
2. Perform all the operation of double ended queue using arrays
3. Display the content of queue at last.

Double ended queue ADT using doubly linked list

1. Create a linked list
2. Perform all the operation of double ended queue using linked list
3. Display the content of queue at last.

4.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

4.5 SOURCE CODE:

Programs to implement a double ended queue ADT using arrays

```
#include<stdio.h>
#define SIZE 30
int dequeue[SIZE];
int front=-1,rear=-1; /* initializing front and rear*/
void insertrear(int);
void deletefront();
void insertfront(int);
void deleterear();
void traverse();
/*main program*/
void main()
{
      int choice,item;
      char ch;
      do
     {
         printf("\n options are");
         printf("\n press 1 to insert at rear");
         printf("\n press 2 to delete at front");
         printf("\n press 3 to insert at front");
         printf("\n press 4 to delete at rear");
         scanf("%d",&choice);
         switch(choice)    /*switch case*/
        {
             case 1: printf("\n enter the element:");
                     scanf("%d",&item);
                     insertrear(item); /*function call for inserting element at rear*/
                     break;
             case 2: deletefront(); /*function call for deleting element at front*/
```

```
                    break;
              case 3: printf("enter the element:");
                    scanf("%d",&item);
                    insertfront(item); /*function call for inserting element at
                    front*/ break;
             case 4:  deleterear();/*function call for deleting element at rear*/
                     break;
              case 5: traverse(); /*traversing the list*/
                     break;
             default : printf("wrong choice");
            } /*end of switch case*/
           printf("\n do you want to perform more operations?(Y/N):");
           fflush(stdin);
           scanf(" %c",&ch);
    } while(ch=='Y'||ch=='y');
  }

/*insertion at rear*/
void insertrear(int value) /*function definition*/
{
   if(rear==(SIZE-1))
   {
       printf("overflow");
        return;
    }
   else
   {
       if(front==-1)
      {
           printf("underflow so front will be modified");
          front=front+1;
      }
       rear=rear+1;
       dequeue[rear]=value;
    }
}

/*deletion at front*/
void deletefront() /*function definition*/
{
    int value;
    if(front==-1)
   {
        printf("queue is already empty");
        value=-1;
   }
   else
   {
      value=dequeue[front];
      if(front==rear)
     {
         printf("queue contains only one item");
         rear=-1;
         front=-1;
     }
     else
     front=front+1;
   }
```

```
   printf("removed element from front is %d",value);
}

/*insertion at front*/
void insertfront(int value)  /*function definition*/
{
    if(front==0)
   {
       printf("front is at the beginning");
      printf("here insertion is not possible");
      return;
   }
   else
  {
     if(front==-1)
   {
       printf("queue is empty so both pointers will modified");
      front=front+1;
      rear=rear+1;
    }
    else
   {
     front=front-1;
   }
   dequeue[front]=value;
 }
}

/*deletion at rear*/
void deleterear()  /*function definition*/
{
    int value;
   if(front==-1)
   {
           printf("queue is already empty");
           return;
   }
  else
 {
      value=dequeue[rear];
      if(rear==front)
     {
       printf("queue contains only one item");
       printf("rear and front will be modified");
       rear=-1;
       front=-1;
     }
     else
    {
      rear=rear-1;
     }
 }
   printf("\n the removed element from rear is:%d",value);
}
/*traverse operation*/
void traverse() /*function definition*/
{
     int i;
```

```
            if(front==-1)
            {
               printf("queue empty");
               return;
            }
           else
           {
               printf("\n value in the queue are as follow:");
               for(i=front;i<=rear;i++)
              printf("\n%d",dequeue[i]);
            }
         }
```

program to implement double ended queue adt using doubly linked list

```
#include <stdio.h>
 #include <stdlib.h>
/*declaring a structure to create a node*/

  struct node
{
      int data;
      struct node *prev, *next;
  };

  struct node *head = NULL, *tail = NULL;

  struct node * createNode(int data)
{
/*allocating implicit memory to the node*/

      struct node *newnode = (struct node *)malloc(sizeof (struct node));
      newnode->data = data;
      newnode->next = newnode->prev = NULL;
      return (newnode);
  }

   /* create sentinel(dummy head & tail) that helps us to do insertion and deletion
    operation at front and rear so easily. And these dummy head and tail wont get deleted
    till the end of execution of this program */

  void createSentinels() /*creating a head and tail*/ {

      head = createNode(0);
      tail = createNode(0);
      head->next = tail;
      tail->prev = head;
  }

  /* insertion at the front of the queue */
  void enqueueAtFront(int data)
 {
      struct node *newnode, *temp;
      newnode = createNode(data);
      temp = head->next;
      head->next = newnode;
      newnode->prev = head;
      newnode->next = temp;
      temp->prev = newnode;
```

```
}

/*insertion at the rear of the queue */
void enqueueAtRear(int data)
{
    struct node *newnode, *temp;
    newnode = createNode(data);
    temp = tail->prev;
    tail->prev = newnode;
    newnode->next = tail;
    newnode->prev = temp;
    temp->next = newnode;
}

/* deletion at the front of the queue */
void dequeueAtFront()
{
    struct node *temp;
    if (head->next == tail)
    {
        printf("Queue is empty\n");
    }
    Else
    {
        temp = head->next;
        head->next = temp->next;
        temp->next->prev = head;
        free(temp);
     }
     return;
}


/* deletion at the rear of the queue */

void dequeueAtRear()
{
    struct node *temp;
    if (tail->prev == head)
    {
        printf("Queue is empty\n");
    }
    Else
     {
        temp = tail->prev;
        tail->prev = temp->prev;
        temp->prev->next = tail;
        free(temp);
    }
    return;
}

/* display elements present in the queue */
void display()
{
    struct node *temp;
    if (head->next == tail)
     {
```

```
            printf("Queue is empty\n");
            return;
        }

        temp = head->next;
        while (temp != tail)
        {
            printf("%-3d", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }

/*main program*/
    int main()
    {
        int data, ch;
        createSentinels();
        while (1)
        {
            printf("1. Enqueue at front\n2. Enqueue at rear\n");
            printf("3. Dequeue at front\n4. Dequeue at rear\n");
            printf("5. Display\n6. Exit\n");
            printf("Enter your choice:");
            scanf("%d", &ch);
            switch (ch) /*switch case*/
            {
                case 1:
                    printf("Enter the data to insert:");
                    scanf("%d", &data);
                    enqueueAtFront(data);
                    break;

                case 2:
                    printf("Enter ur data to insert:");
                    scanf("%d", &data);
                    enqueueAtRear(data);
                    break;

                case 3:
                    dequeueAtFront();
                    break;

                case 4:
                    dequeueAtRear();
                    break;

                case 5:
                    display();
                    break;

                case 6:
                    exit(0);

                default:
                    printf("Pls. enter correct option\n");
                    break;
            } /*end of switch case*/
```

```
    }
    return 0;
}
```

4.6 PRE LAB QUESTIONS

1. what is queue and its operations
2. what is double ended queue
3. differentiate queue and double ended queue

4.7 LAB ASSIGNMENT

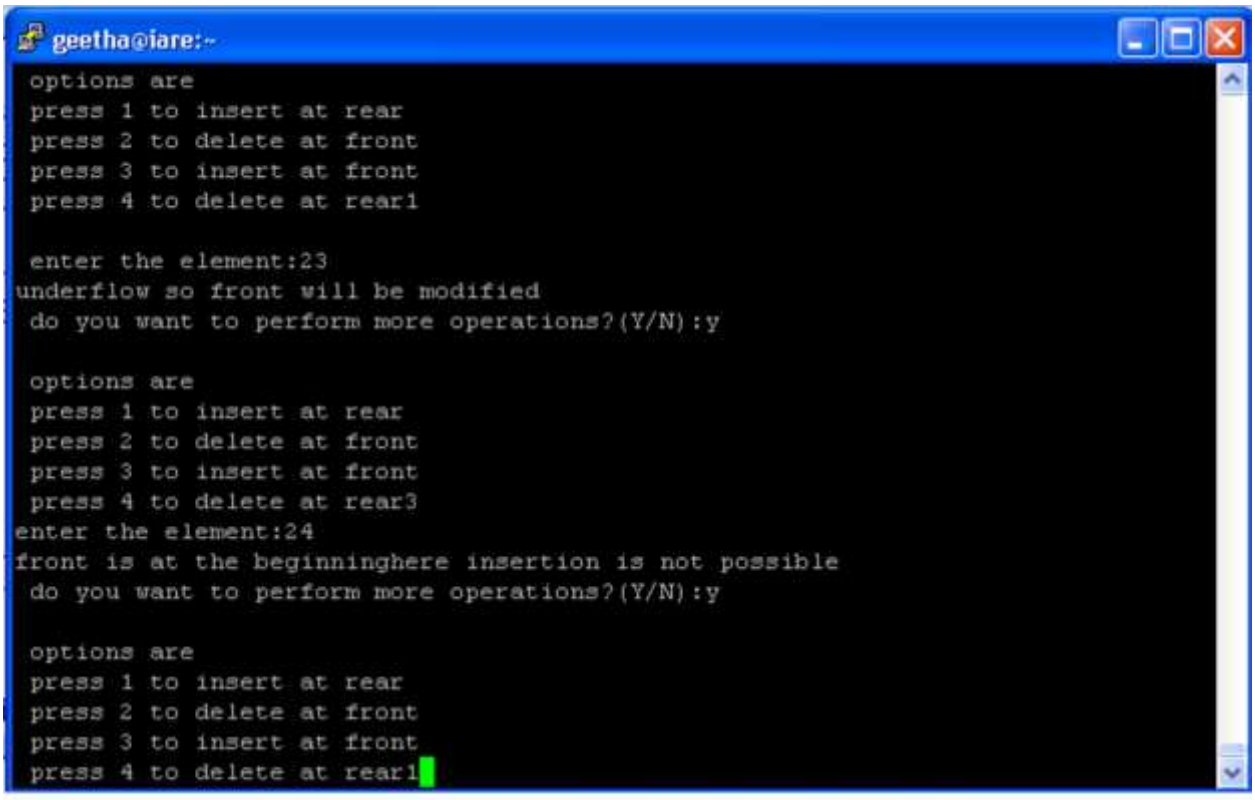
1. Write a program to insert an element when rear is at last position
2. Write a program to delete an element when front is at last position

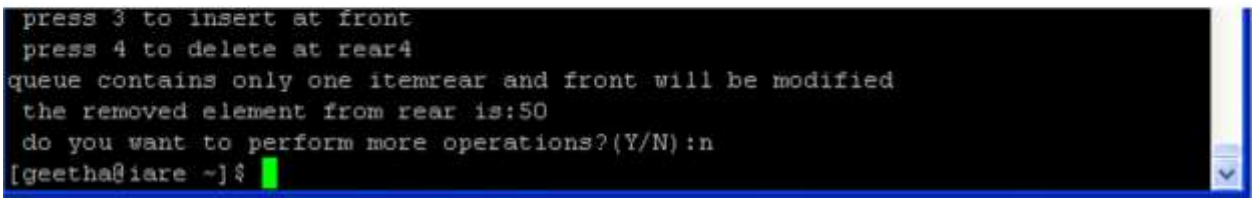
4.8 POST LAB QUESTIONS

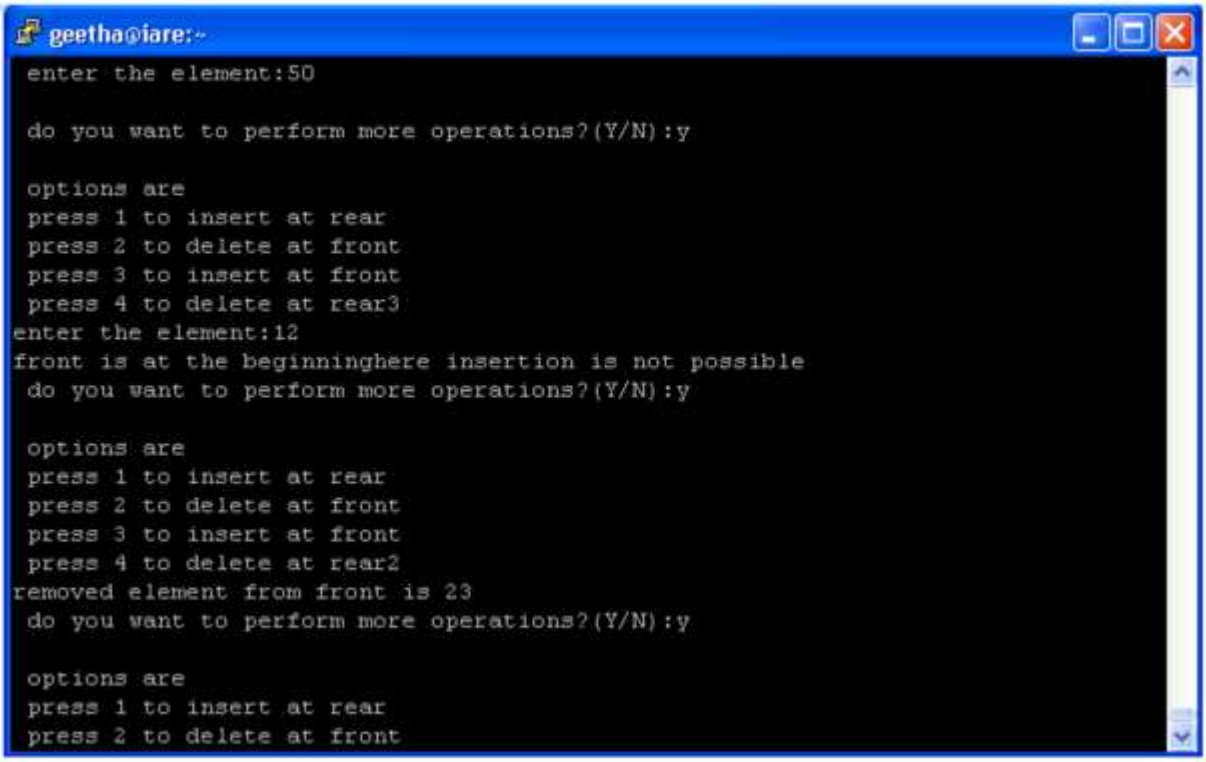
1. Write the condition for queue full
2. Write the condition for queue empty
3. List the advantages of double ended queue over queue

4.9 INPUT AND OUTPUT

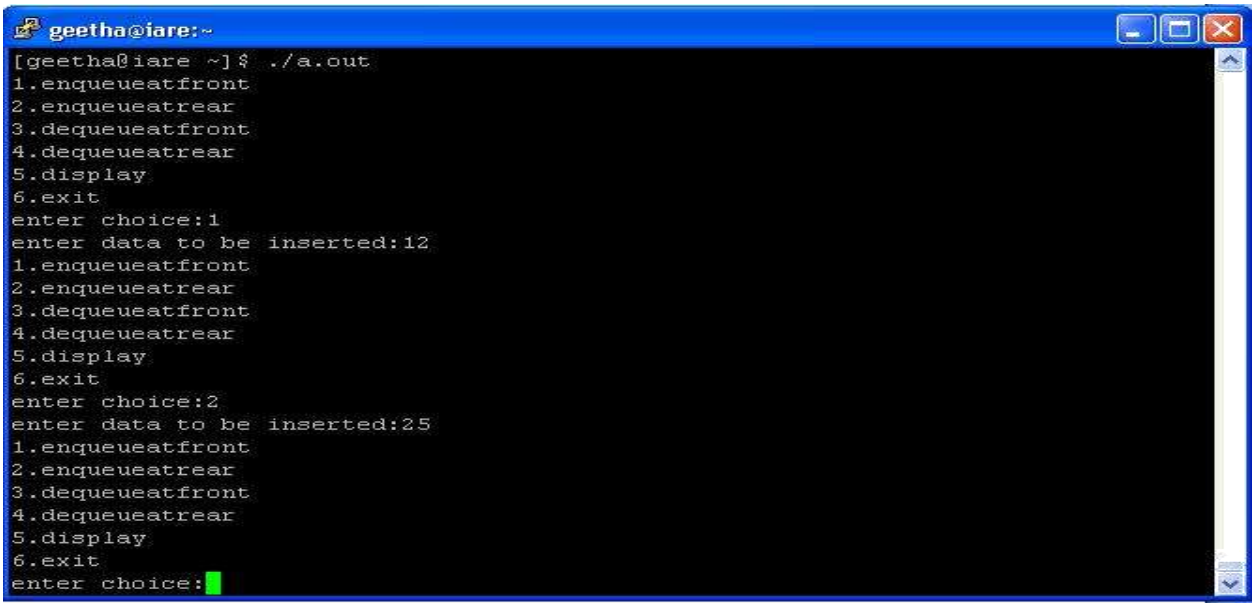
A double ended queue ADT using arrays

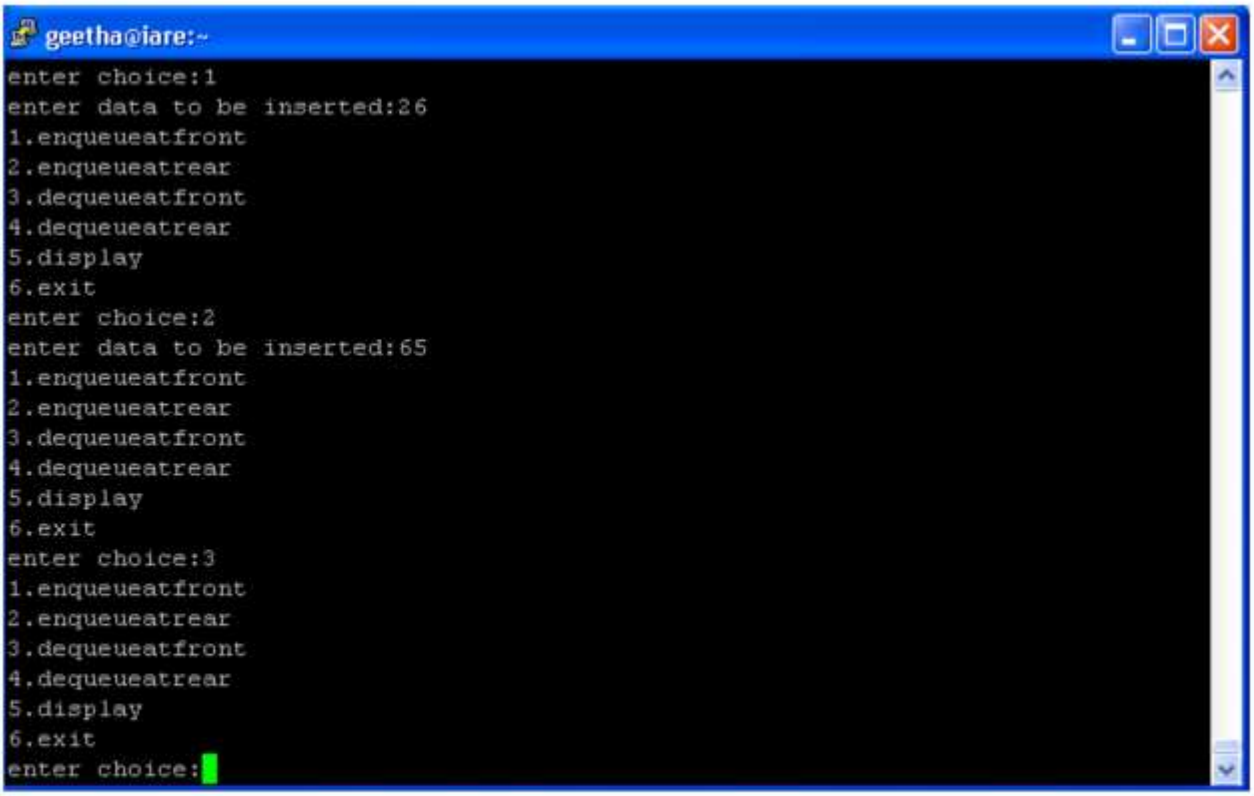

geetha@iare:~
options are
press 1 to insert at rear
press 2 to delete at front
press 3 to insert at front
press 4 to delete at rear1
enter the element:23
underflow so front will be modified
do you want to perform more operations?(Y/N):y
options are
press 1 to insert at rear
press 2 to delete at front
press 3 to insert at front
press 4 to delete at rear3
enter the element:24
front is at the beginninghere insertion is not possible
do you want to perform more operations?(Y/N):y
options are
press 1 to insert at rear
press 2 to delete at front
press 3 to insert at front
press 4 to delete at rear1


press 3 to insert at front
press 4 to delete at rear4
queue contains only one itemrear and front will be modified
the removed element from rear is:50
do you want to perform more operations?(Y/N):n
[geetha@iare ~]$


geetha@iare:~
enter the element:50
do you want to perform more operations?(Y/N):y
options are
press 1 to insert at rear
press 2 to delete at front
press 3 to insert at front
press 4 to delete at rear3
enter the element:12
front is at the beginninghere insertion is not possible
do you want to perform more operations?(Y/N):y
options are
press 1 to insert at rear
press 2 to delete at front
press 3 to insert at front
press 4 to delete at rear2
removed element from front is 23
do you want to perform more operations?(Y/N):y
options are
press 1 to insert at rear
press 2 to delete at front

double ended queue adt using doubly linked list




geetha@iare:~
enter choice:1
enter data to be inserted:26
1.enqueueatfront
2.enqueueatrear
3.dequeueatfront
4.dequeueatrear
5.display
6.exit
enter choice:2
enter data to be inserted:65
1.enqueueatfront
2.enqueueatrear
3.dequeueatfront
4.dequeueatrear
5.display
6.exit
enter choice:3
1.enqueueatfront
2.enqueueatrear
3.dequeueatfront
4.dequeueatrear
5.display
6.exit
enter choice:

enter choice:4
1.enqueueatfront
2.enqueueatrear
3.dequeueatfront
4.dequeueatrear
5.display
6.exit
enter choice:5
12 25
1.enqueueatfront
2.enqueueatrear
3.dequeueatfront
4.dequeueatrear
5.display
6.exit
enter choice:6
[geetha@iare ~]$

EXPERIMENT 5

5.1 OBJECTIVE
1. To create a binary search tree of characters.
2. Traverse the above Binary search tree recursively in Post order.

5.2 RESOURCE:
Turbo C

5.3 PROGRAM LOGIC
1. Create binary tree with the property binary search tree
2. Visit the tree in post order
3. Visit in the order left, right, root
4. Display the visited nodes

5.4 PROCEDURE:
Go to debug -> run or press CTRL + F9 to run the program

5.5 SOURCE CODE:

/*program for creating and traversing the binary search tree*/

```
#include<stdio.h>
#include<stdlib.h>
typedef struct BST
{
    char d;
/*declaring a structure to create a node*/

   struct BST *lc,*rc;
}node;
/*main program*/
void main()
{
   int choice;
   char ans='N';
    int key;
    node *nn,*root,*parent;
    root=NULL;
    printf("\n program for binary search tree");
    do
   {
     printf("\n 1.create");
     printf("\n 2.resurcive traverse");
     printf("\n 3.exit");
     printf("\n enter your choice");
     scanf("%d",&choice);
     switch(choice) /*switch case*/
     {
       case 1:
                        do
                        {
                                nn=(node *)malloc(sizeof(node));
                                printf("\n enter the elements");
                                nn->lc=NULL;
```

```
                                nn->rc=NULL;
                                scanf(" %c",&nn->d);
                                if(root==NULL)
                                  root=nn;
                                else
                                    insert(root,nn);
                                printf("\n want to enter more elements?(Y/N)");
                                scanf(" %c",&ans);
                            }while(ans=='y');
                            break;
        case 2:
                            if(root==NULL)
                            printf("tree is not created");
                            else
                            {
                                printf("\n the inorder display:");
                                inorder(root);
                                printf("\n the preorder display:");
                                preorder(root);
                                printf("\n the postorder display:");
                                postorder(root);
                            }
                            break;
     } /*end of switch case*/
    }while(choice!=3);
 }
/*insertion operation*/
void insert(node *root,node *nn)
{
   int c,d;
   c=nn->d;
   d=root->d;
   if(c<d)
   {
     if(root->lc==NULL)
     root->lc=nn;
    else
      insert(root->lc,nn);
   }
}
/*inorder traversal*/
void inorder(node *temp)
{
   if(temp!=NULL)
   {
     inorder(temp->lc);
     printf(" %c",temp->d);
     inorder(temp->rc);
   }
}
/*preorder traversal*/
void preorder(node *temp)
{
  if(temp!=NULL)
  {
    printf(" %c",temp->d);
    preorder(temp->lc);
    preorder(temp->rc);
```

```
  }
}
/*postorder traversal*/
void postorder(node *temp)
{
  if(temp!=NULL)
  {
    postorder(temp->lc);
    postorder(temp->rc);
    printf(" %c",temp->d);
  }
}
```

5.6 PRE LAB QUESTIONS

1. Differentiate between BST and complete BST
2. What are the properties of BST
3. How many nodes will be there in given nth level.

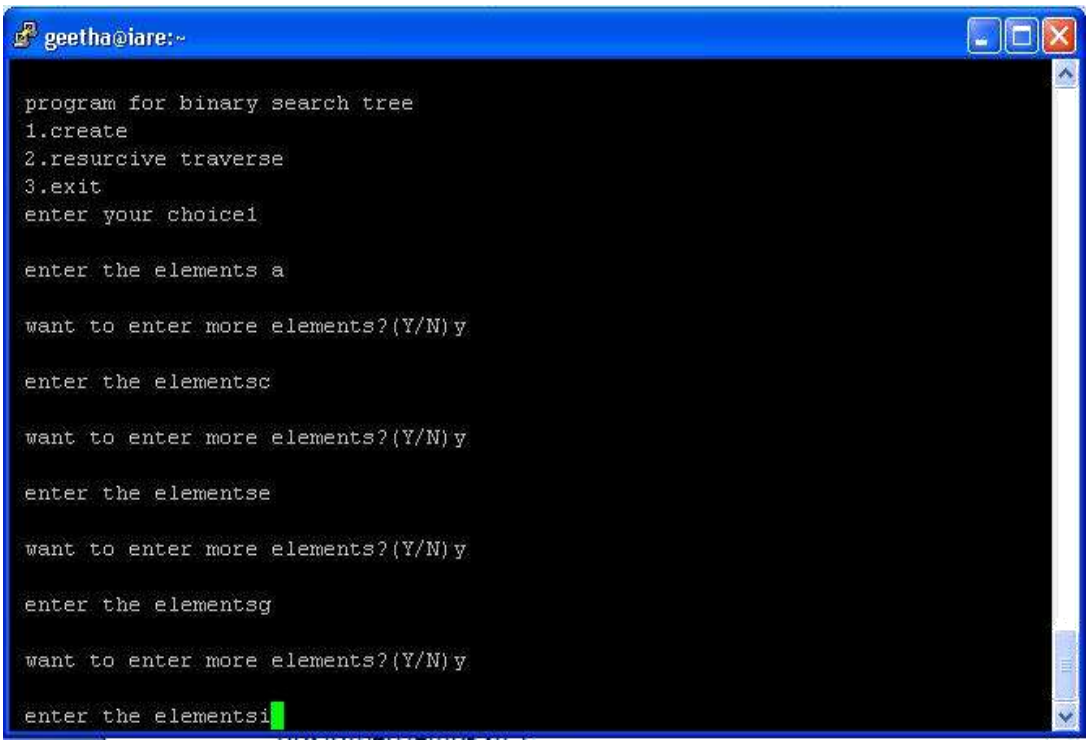
5.7 LAB ASSIGNMENT

1. Construct a binary search tree for the following 80, 40, 75, 30, 20, 90, 50

5.8 POST LAB QUESTIONS

1. List the various tree traversal techniques are there
2. Write the necessary condition for inserting element into BST
3. List the applications of BST

5.9 INPUT AND OUTPUT



EXPERIMENT 6

6.1 OBJECTIVE
1. Create a binary search tree of integers.
2. Traverse the above Binary search tree non recursively in inorder.

6.2 RESOURCE:
Turbo C

6.3 PROGRAM LOGIC
1. Read integers
2. Create binary tree with the property binary search tree
3. Visit the tree in inorder
4. Visit in the order left, root, right,
5. Display the visited nodes

6.4 PROCEDURE
Go to debug -> run or press CTRL + F9 to run the program

6.5 SOURCE CODE:

```
/*creating binary search tree of integer*/

#include<stdio.h>
#include<conio.h>
typedef struct bint
{
/*declaring a structure to create a node*/

  int data,flag;
  struct bint *left,*right;
}node;
node * create(node *r,int d)
{
   if(r == NULL)
   {
/*allocating implicit memory to the node*/

      r = (node *)malloc(sizeof(node));
      r->data = d;
      r->left = r->right = NULL;
  }
  else
 {
   if(r->data <= d)
       r->right = create(r->right,d);
   else
       r->left = create(r->left,d);
 }
 return r;
}
void non_in(node *r)
{
   int top=0;
   node *s[20],*pt=r;
   s[0]=NULL;
```

```
    while(pt != NULL)
    {
      s[++top] = pt;
      pt = pt->left;
    }
    pt = s[top--];
    while(pt != NULL)
    {
       printf("%d\t",pt->data);
       if(pt->right != NULL)
       {
         pt = pt->right;
         while(pt != NULL)
         {
           s[++top] = pt;
           pt = pt->left;
         }
    }
    pt = s[top--];
  }
}
/*main program*/
void main()
{
     int d;
    char ch = 'Y';
     node *root = NULL;
    clrscr();
    while(toupper(ch) == 'Y')
    {
      printf("\n Enter the item to insert");
      scanf("%d",&d);
      head = create(root,d);
      printf("\n Do you want to continue(y/n)");
      fflush(stdin);
      ch = getchar();
    }

    printf("\n inorder non recursive\n");
    non_in(head);
}
```

6.6 PRE LAB QUESTIONS

1. Differentiate between BST and complete BST
2. What are the properties of BST
3. How many nodes will be there in given nth level.

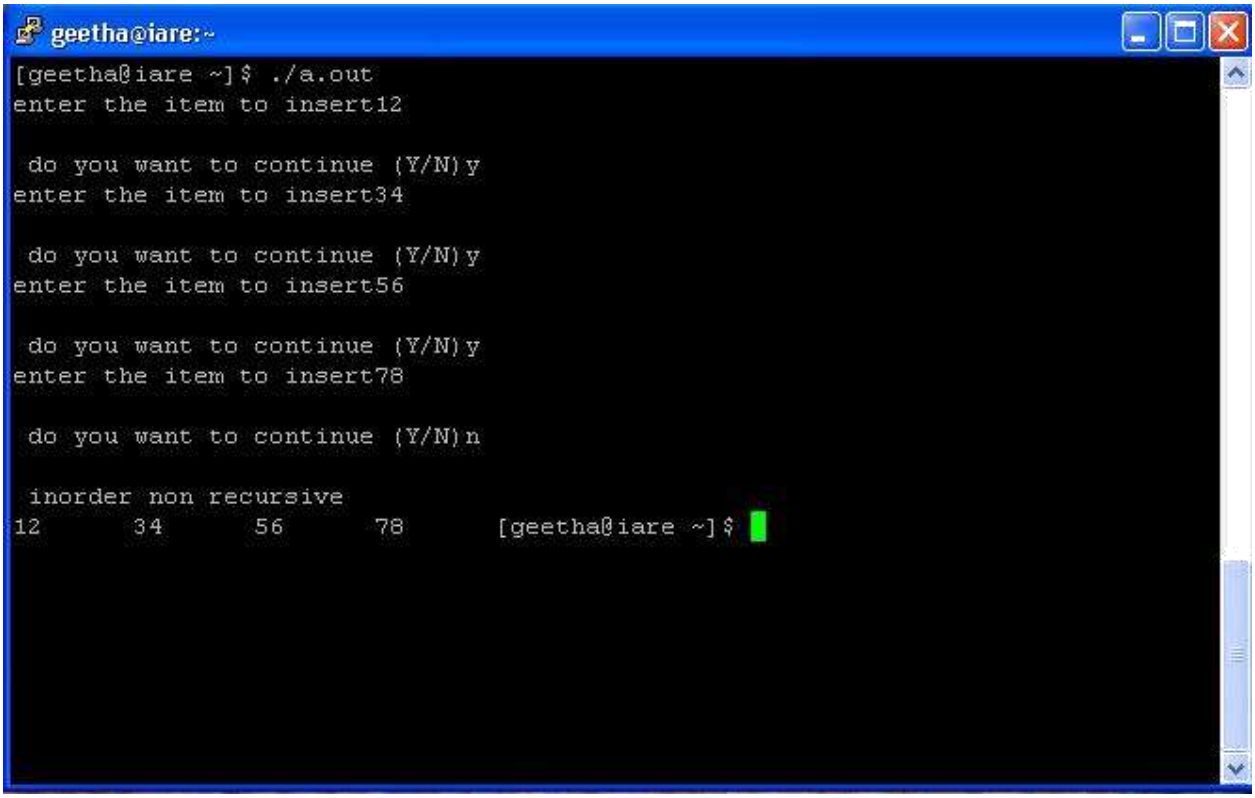
6.7 LAB ASSIGNMENT

1 Construct a binary search tree for the following 80, 40, 75, 30, 20, 90, 50

6.8 POST LAB QUESTIONS

1. List the various tree traversal techniques are there
2. Write the necessary condition for inserting element into BST
3. List the applications of BST

6.9 INPUT AND OUTPUT


geetha@iare:~
[geetha@iare ~]$ ./a.out
enter the item to insert12
do you want to continue (Y/N)y
enter the item to insert34
do you want to continue (Y/N)y
enter the item to insert56
do you want to continue (Y/N)y
enter the item to insert78
do you want to continue (Y/N)n
inorder non recursive
12 34 56 78 [geetha@iare ~]$

EXPERIMENT 7

7.1 OBJECTIVE

1. Arrange a list of integers in ascending order using Insertion sort
2. Arrange a list of integers in ascending order using Merge sort

7.2 RESOURCE

Turbo C

7.3 PROGRAM LOGIC

INSERTION SORT

1. Start with an empty left hand [sorted array] and the cards face down on the table [unsorted array].
2. Then remove one card [key] at a time from the table [unsorted array], and insert it into the correct position in the left hand [sorted array].
3. To find the correct position for the card, we compare it with each of the cards already in the hand, from right to left.

MERGE SORT

1. **Divide Step**
   If a given array *A* has zero or one element, simply return; it is already sorted. Otherwise, split *A*[*p* .. *r*] into two subarrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*], each containing about half of the elements of *A*[*p* .. *r*]. That is, *q* is the halfway point of *A*[*p* .. *r*].
2. **Conquer Step**
   Conquer by recursively sorting the two subarrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*].
3. **Combine Step**
   Combine the elements back in *A*[*p* .. *r*] by merging the two sorted subarrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (*A*, *p*, *q*, *r*).

7.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

7.5 SOURCE CODE:

Insertion sort

```
#include<stdio.h>
void inst_sort(int[]);
void main()  {
        int num[5],count;
        printf("\n enter the five elements to sort:\n");
        for(count=0;count<5;count++)
          scanf("%d",&num[count]);
        inst_sort(num)**; /*function call for insertion sort*/**
        printf("\n\n elements after sorting:\n");
        for(count=0;count<5;count++)
          printf("%d\n",num[count]);
}
void inst_sort(int num[])  {   **/* function definition for insertion sort*/**
     int i,j,k;
     for(j=1;j<5;j++)      {
       k=num[j];
       for(i=j-1;i>=0&&k<num[i];i--)
```

```
        num[i+1]=num[i];
        num[i+1]=k;
    }}
```

Merge sort

```
#include<stdio.h>
void mergesort(int[],int,int);
void mergearray(int[],int,int,int);
main()  {
     int a[50],n,i;
    printf("\n enter size of an array:");
    scanf("%d",&n);
     printf("\n enter elements of an array:\n");
     for(i=0;i<n;i++)
       scanf("%d",&a[i]);
     mergesort(a,0,n-1);
     printf("\n\nafter sorting:\n");
     for(i=0;i<n;i++)
       printf("\n%d",a[i]);
}
/*merge operation*/
void mergesort(int a[],int beg,int end)  {
   int mid;
   if(beg<end)    {
      mid=(beg+end)/2;
     mergesort(a,beg,mid);
     mergesort(a,mid+1,end);
     mergearray(a,beg,mid,end);
  } }
void mergearray(int a[],int beg,int mid,int end)  {
    int i,leftend,num,temp,j,k,b[50];
    for(i=beg;i<=end;i++)
    b[i]=a[i];
    i=beg;
    j=mid+1;
    k=beg;
    while((i<=mid)&&(j<=end))     {
   if(b[i]<+b[j])
   {
     a[k]=b[i];
     i++;
     k++;
   }
  else    {
     a[k]=b[j];
     j++;
     k++;
   } }

 if(i<=mid)  {
    while(i<=mid)     {
      a[k]=b[i];
      i++;
      k++;
   } }
  else  {
    while(j<=end)  {
```

```
        a[k]=b[j];
        j++;
        k++;
    } } }
```

7.6 PRE LAB QUESTIONS

1. what is sorting
2. List the various sorting algorithms
3. What is the advantage of insertion sort
4. How merge sort works
5. Is insertion sort is stable algorithm

7.7 LAB ASSIGNMENT

1. Apply the selection sort on the following elements21,11,5,78,49, 54,72,88
2. Apply the merge sort on the following elements21,11,5,78,49, 54,72,88 and 56,28,10

7.8 POST LAB QUESTIONS

1. What is the time complexity of insertion sort
2. What is the time complexity of merge sort
3. Why sorting is required
4. Is merge sort is in place
5. List the application of merge sort

7.9 INPUT AND OUTPUT

Insertion sort

```
geetha@iare:~
[geetha@iare ~]$ gcc insert.c
[geetha@iare ~]$ ./a.out

 enter the five elements to sort:
5 4 3 2 1


 elements after sorting:
1
2
3
4
5
[geetha@iare ~]$
```

Merge sort

```
geetha@iare:~
[geetha@iare ~]$ gcc week7b.c
[geetha@iare ~]$ ./a.out

 enter size of an array:5

 enter elements of an array:
5 4 3 2 1


after  sorting:

1
2
3
4
5[geetha@iare ~]$
```

EXPERIMENT 8

8.1. OBJECTIVE

1. Sorting the list of integers in ascending order using Quick sort
2. Sorting the list of integers in ascending order using Selection sort

8.2 RESOURCE:

Turbo C

8.3 PROGRAM LOGIC

Quick sort

1. Read the elements to be sort
2. Find the proper pivot element
3. Apply quick sort method to sort the remaining elements

Selection sort

1. Read the elements to be sort
2. Select the minimum element
3. Apply the selection sort to sort the remaining elements

8.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

8.5 SOURCE CODE:

QUICK SORT

```
#include<stdio.h>
main()
{
    int x[10],i,n;
   printf("enter number of elements:");
   scanf("%d",&n);
   printf("enter %d elements:\n");
   for(i=0;i<n;i++)
    scanf("%d",&x[i]);
   quicksort(x,0,n-1);/*function call*/
   printf("sorted elements are:");
   for(i=0;i<n;i++)
     printf("%3d",x[i]);
}
/*called function*/
quicksort(int x[10],int first,int last)
{
    int pivot,i,j,t;
    if(first<last)
   {
      pivot=first;
       i=first;
       j=last;
      while(i<j)
      {
        while(x[i]<=x[pivot]&&i<last)
        i++;
       while(x[j]>x[pivot])
        j--;
      if(i<j)
```

```
      {
        t=x[i];
        x[i]=x[j];
        x[j]=t;
      }
    }
     t=x[pivot];
     x[pivot]=x[j];
     x[j]=t;
     quicksort(x,first,j-1);
     quicksort(x,j+1,last);
   }
}
```

SELECTION SORT

```
#include<stdio.h>
void sel_sort(int[]);
void main()
{
    int num[5],count;
   printf("enter the five elements to sort:\n");
   for(count=0;count<5;count++)
     scanf("%d",&num[count]);
   sel_sort(num); /*function call*/
    printf("\n\n elements after sorting:\n");
   for(count=0;count<5;count++)
     printf("%d\n",num[count]);
}
/*called function*/
void sel_sort(int num[])
{
   int i,j,min,temp;
   for(j=0;j<5;j++)
  {
     min=j;
     for(i=j;i<5;i++)
     if(num[min]>num[i])
     min=i;
    if(min<5)
    {
        temp=num[j];
        num[j]=num[min];
        num[min]=temp;
    }
     printf("%d\t",num[j]);
  }
}
```

8.6 PRE LAB QUESTIONS

1. what is sorting
2. List the various sorting algorithms
3. What is the advantage of selection sort
4. How to find pivot element in quick sort
5. Is quick sort is stable algorithm

8.7 LAB ASSIGNMENT

1 . Rearrange the following numbers using Quick sort procedure. 42, 12, 18, 98, 67, 83, 8, 10, 71

2.Apply the selection sort on the following elements21,11,5,78,49, 54,72,88

8.8 POST LAB QUESTIONS

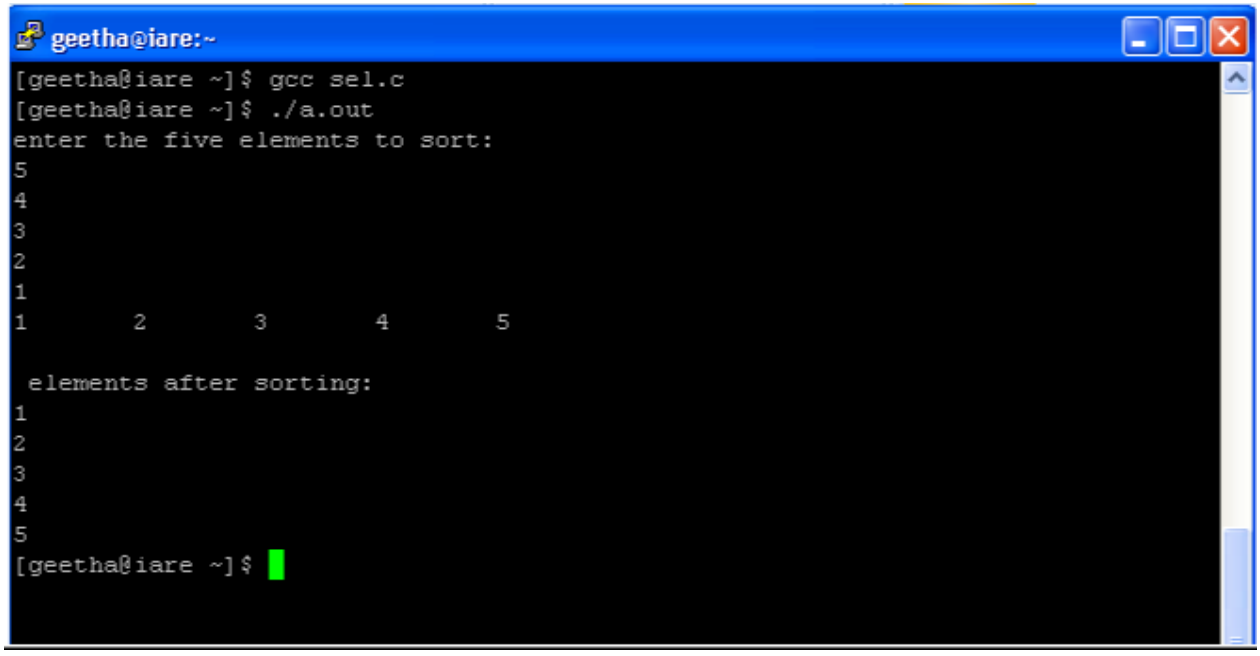
6. What is the time complexity of selection sort
7. What is the time complexity of quick sort
8. Why sorting is required
9. Is selection sort is stable
10. What is the worst case for quick sort

8. 9 INPUT AND OUTPUT

QUICK SORT

geetha@iare:~
[geetha@iare ~]$ gcc quick.c
[geetha@iare ~]$ ./a.out
enter number of elements:5
enter 1 elements:
5 4 3 2 1
sorted elements are: 1 2 3 4 5[geetha@iare ~]$

SELECTION SORT



EXPERIMENT 9

9.1 OBJECTIVE

1. To perform the Insertion into a B-tree
2. Implement Heap sort algorithm

9.2 RESOURCE:

Turbo C

9.3 PROGRAM LOGIC

B-tree

1. Read the elements
2. Use the properties B-Tree and construct B-Tree
3. Select the required operation

Heap sort

4. Read the elements to be sort
5. Construct the heap
6. Sort the above heap using deletion

9.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

9.5 SOURCE CODE:

B-TREE

```
#include<stdio.h>
#include <stdlib.h>
#define M 5
struct node
{
int n; /* n < M No. of keys in node will always less than order of B
                tree */
int keys[M-1];      /*array of keys*/
struct node *p[M]; /* (n+1 pointers will be in use) */
}*root=NULL;

enum KeyStatus { Duplicate,SearchFailure,Success,InsertIt,LessKeys
}; void insert(int key);
void display(struct node *root,int);
void search(int x);
enum KeyStatus ins(struct node *r, int x, int* y, struct node** u);
int searchPos(int x,int *key_arr, int n); int main()

{
int key;
int choice;
printf("Creation of B tree for node %d\n",M);
while(1)
{
printf("1.Insert\n");
printf("3.Search\n");
printf("4.Display\n");
printf("5.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);
```

```
switch(choice)
{
        case 1:
                printf("Enter the key : ");
                scanf("%d",&key);
                insert(key);
                break;


        case 3:
                printf("Enter the key : ");
                scanf("%d",&key);
                search(key);
                break;
        case 4:
                printf("Btree is :\n");
                display(root,0);
                break;
        case 5:
                exit(1);
        default:
                printf("Wrong choice\n");
                break;
        }/*End of switch*/
        }/*End of while*/
return 0;
}/*End of main()*/
void insert(int key)
{
struct node *newnode;
int upKey;
enum KeyStatus value;
value = ins(root, key, &upKey, &newnode);
if (value == Duplicate)
printf("Key already available\n");
if (value == InsertIt)
{
struct node *uproot = root;
root=malloc(sizeof(struct node));
root->n = 1;
root->keys[0] = upKey;
root->p[0] = uproot;
root->p[1] = newnode;
}/*End of if */
}/*End of insert()*/

enum KeyStatus ins(struct node *ptr, int key, int *upKey,struct node**newnode)
{
struct node *newPtr, *lastPtr;
int pos, i, n,splitPos;
int newKey, lastKey;
enum KeyStatus value;
if (ptr == NULL)
{
*newnode = NULL;
*upKey = key;
return InsertIt;
```

```
}
n = ptr->n;
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
return Duplicate;
value = ins(ptr->p[pos], key, &newKey, &newPtr);
if (value != InsertIt)
return value;
/*If keys in node is less than M-1 where M is order of B tree*/
if (n < M - 1)
{
pos = searchPos(newKey, ptr->keys, n);
/*Shifting the key and pointer right for inserting the new key*/
for (i=n; i>pos; i--)
{
ptr->keys[i] = ptr->keys[i-1];
ptr->p[i+1] = ptr->p[i];
}
/*Key is inserted at exact location*/
ptr->keys[pos] = newKey;
ptr->p[pos+1] = newPtr;
++ptr->n; /*incrementing the number of keys in node*/
        return Success;
}/*End of if */
/*If keys in nodes are maximum and position of node to be inserted
is last*/
if (pos == M - 1)
{
lastKey = newKey;
lastPtr = newPtr;
}
else /*If keys in node are maximum and position of node to be inserted
is not last*/
{
lastKey = ptr->keys[M-2];
lastPtr = ptr->p[M-1];
for (i=M-2; i>pos; i--)
{
ptr->keys[i] = ptr->keys[i-1];
ptr->p[i+1] = ptr->p[i];
}
ptr->keys[pos] = newKey;
ptr->p[pos+1] = newPtr;
}
splitPos = (M - 1)/2;
(*upKey) = ptr->keys[splitPos];

(*newnode)=malloc(sizeof(struct node));/*Right node after split*/
ptr->n = splitPos; /*No. of keys for left splitted node*/
(*newnode)->n = M-1-splitPos;/*No. of keys for right splitted node*/
for (i=0; i < (*newnode)->n; i++)
{
(*newnode)->p[i] = ptr->p[i + splitPos + 1];
if(i < (*newnode)->n - 1)
(*newnode)->keys[i] = ptr->keys[i + splitPos + 1];
else
(*newnode)->keys[i] = lastKey;
}
```

```
(*newnode)->p[(*newnode)->n] = lastPtr;
return InsertIt;
}/*End of ins()*/

void display(struct node *ptr, int blanks)
{
if (ptr)
{
int i;
for(i=1;i<=blanks;i++)
printf(" ");
for (i=0; i < ptr->n; i++)
printf("%d ",ptr->keys[i]);
printf("\n");
for (i=0; i <= ptr->n; i++)
display(ptr->p[i], blanks+10);
}/*End of if*/
}/*End of display()*/

void search(int key)
{
int pos, i, n;
struct node *ptr = root;
printf("Search path:\n");
while (ptr)
{
n = ptr->n;
for (i=0; i < ptr->n; i++)
printf(" %d",ptr->keys[i]);
printf("\n");
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
{
printf("Key %d found in position %d of last dispalyed node\n",key,i);
return;
}
ptr = ptr->p[pos];
}
printf("Key %d is not available\n",key);
}/*End of search()*/

int searchPos(int key, int *key_arr, int n)
{
int pos=0;
while (pos < n && key > key_arr[pos])
pos++;
return pos;
}/*End of searchPos()*/
```

HEAP SORT

```
#include<stdio.h>
int p(int);
int left(int);
int right(int);
void heapify(int[],int,int);
void buildheap(int[],int);
```

```
void heapsort(int[],int);
void main()
  {
     int x[20],n,i;
     printf("enter the no. of elements to b sorted");
     scanf("%d",&n);
     printf("enter the elements ");
     for(i=0;i<n;i++)
      scanf("%d",&x[i]);
     heapsort(x,n);
     printf("sorted array is");
     for(i=0;i<n;i++)
      printf("%d",x[i]);
   }
  int p(int i)
  {
     return i/2;
  }
  int left(int i)
 {
   return 2*i+1;
  }
  int right(int i)
  {
    return 2*i+2;
  }
  void heapify(int a[],int i,int n)
  {
     int l,r,large,t;
     l=left(i);
     r=right(i);
       if((l<=n-1)&&(a[l]>a[i]))
         large=l;
      else
         large=i;
     if((r<=n-1)&&(a[r]>a[large]))
        large=r;
     if(large!=i)
     {
       t=a[i];
      a[i]=a[large];
      a[large]=t;
      heapify(a,large,n);
     }
 }
void buildheap(int a[],int n)
{
    int i;
   for(i=(n-1)/2;i>=0;i--)
     heapify(a,i,n);
}
void heapsort(int a[],int n)
{
   int i,m,t;
   buildheap(a,n);
    m=n ;
   for(i=n-1;i>=1;i--)
   {
```

```
        t=a[0];
       a[0]=a[i];
       a[i]=t;
       m=m-1;
        heapify(a,0,m);
     }
  }
```

9.6 PRE LAB QUESTIONS

1. what is B –tree
2. What is heap
3. What are the properties of B-tree
4. What is sorting

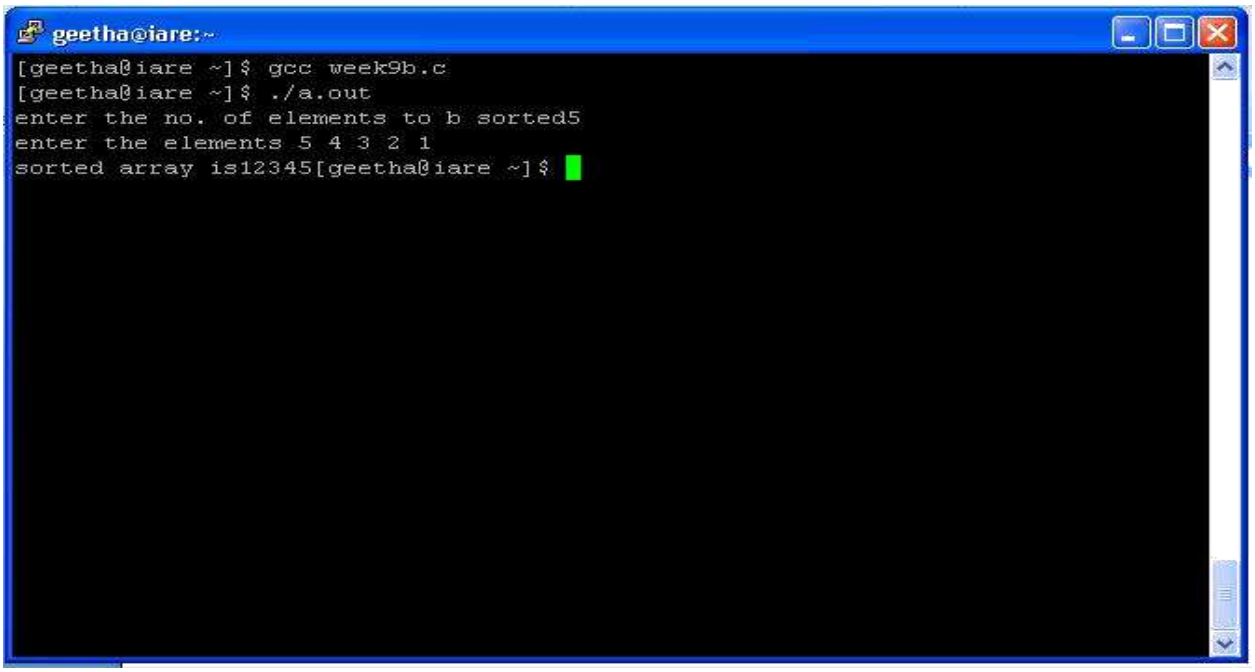
9.7 LAB ASSIGNMENT

1. Apply heap sort on list of elements 14,12,9,8,7,10,18,20,30
2. Construct a B-tree of order 3 with the following elements 10,20,15,3,2,16,21,25,30,40

9.8 POST LAB QUESTIONS

1. What is time complexity of heap sort
2. Write the condition to insert and delete an element into B-tree
3. Is heap sort is stable or not

9. 9 INPUT AND OUTPUT


geetha@iare:~
[geetha@iare ~]$ gcc week9b.c
[geetha@iare ~]$ ./a.out
enter the no. of elements to b sorted5
enter the elements 5 4 3 2 1
sorted array is12345[geetha@iare ~]$

EXPERIMENT 10

10.1 OBJECTIVE

To implement all the functions of a dictionary (ADT) using hashing

10.2 RESOURCE:

Turbo C

10.3 PROGRAM LOGIC

1. Read the key elements from the dictionary
2. Use the hash function to implement dictionary
3. Apply required operation on the dictionary

10.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

10.5 SOURCE CODE:

DICTIONARY USING HASHING

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int b;

int hsearch(int key,int d,int *ht,int *empty)
{
        int i=key%(d);
        int j=i;
        int c=0;
        do
        {
           if(empty[j]||(*(ht+j)==key))
           return j;
           c++;
           j=(i+c)%(d);
        }while(j!=i);
        return 0;
}
int search(int key,int d,int *ht,int *empty)
{
     b=hsearch(key,d,ht,empty);
      printf("%d",b);
      if(empty[b]==1)
        return -1;
     else if(b==0)
        return 1;
     else
      return b;
}
/*insertion operation*/
void insert(int key,int d,int *ht,int *empty)
{
     b=hsearch(key,d,ht,empty);
    if(empty[b])
    {
```

```
            empty[b]=0;
             *(ht+b)=key;
           printf("elements is inserted\n");
       }
}
/*deletion operation*/
void delete(int key,int d,int *ht,int *empty)
{
        int b=hsearch(key,d,ht,empty);
       *(ht+b)=0;
        empty[b]=1;
       printf("element is deleted\n");
}
void display(int d,int *ht,int *empty)
{
    int i;
    printf("hash table elements are\n");
    for(i=0;i<d;i++)
    {
      if(empty[i])
     printf("  0");
    else
      printf("%5d",*(ht+i));
    }
   printf("\n");
 }
/*main program*/
void main()
{
      int choice=1;
      int key;
      int d,i,s;
      int *empty,*ht;
      printf("enter the hash table size:");
      scanf("%d",&d);
      ht=(int *)malloc(d *sizeof(int));
    empty=(int *)malloc(d *sizeof(int));
    for(i=0;i<d;i++)
    empty[i]=1;
    while(1)
    {
           printf("\n");
            printf("\n LINEAR PROBING");
            printf("\n 1:insert hash table:");
            printf("\n 2:delete hash table");
            printf("\n 3:search hash table");
            printf("\n 4:display hash table");
            printf("\n 5:exit");
            printf("enter your choice");
            scanf("%d",&choice);
            switch(choice)
            {
                case 1:printf("enter the elemants:");
                        scanf("%d",&key);
                        insert(key,d,ht,empty);
                        break;
                case 2:printf("enter to remove from hash table:");
                        scanf("%d",&key);
```

```
                    delete(key,d,ht,empty);
                    break;
                case 3:printf("enter the search elements:");
                    scanf("%d",&key);
                     s=search(key,d,ht,empty);
                     if(s==-1||s==0)
                      printf("not found\n");
                    else
                      printf("element found at index
                    %d",hsearch(key,d,ht,empty)); break;
                case 4:display(d,ht,empty);
                      break;
                case 5:exit(0);
            }
        }return;
    }
```

10.6 PRE LAB QUESTIONS

1. what is dictionary
2. What is hashing
3. List types of hash functions
4. Define linear probing
5. What is quadratic probing

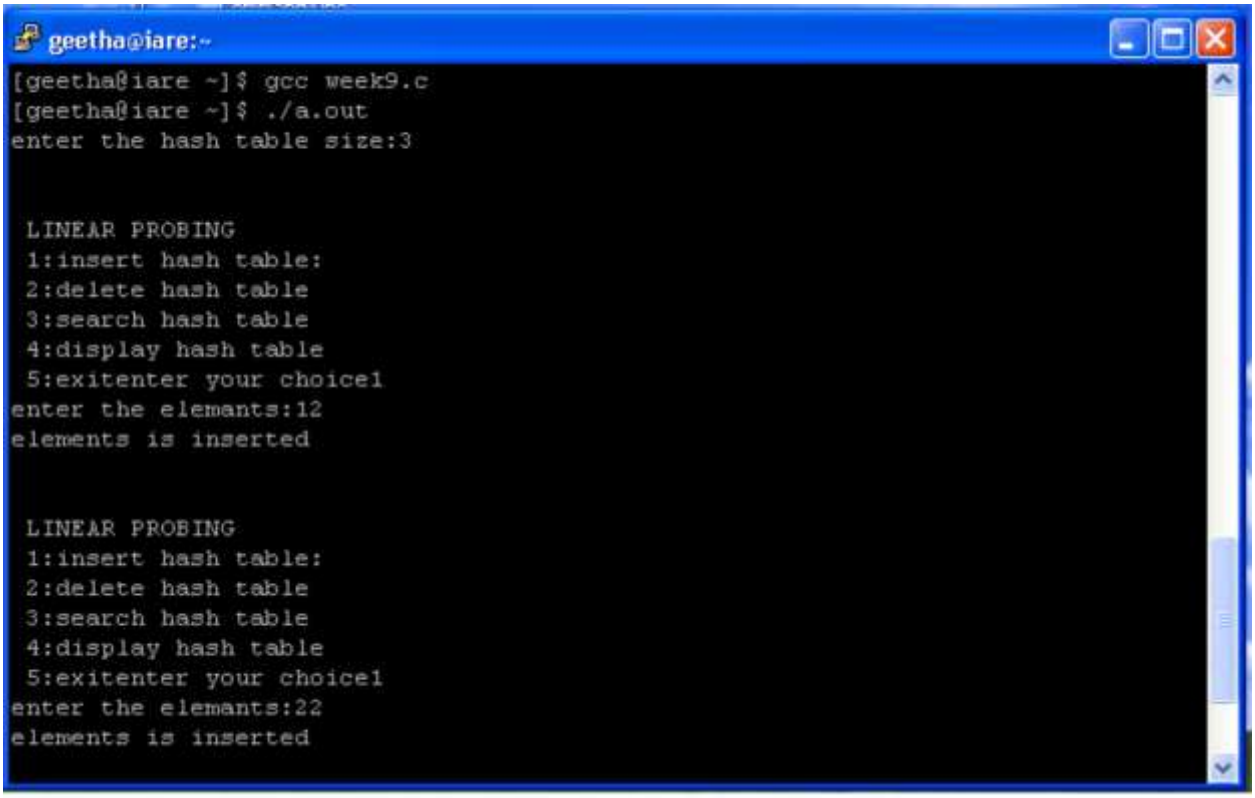
10.7 LAB ASSIGNMENT

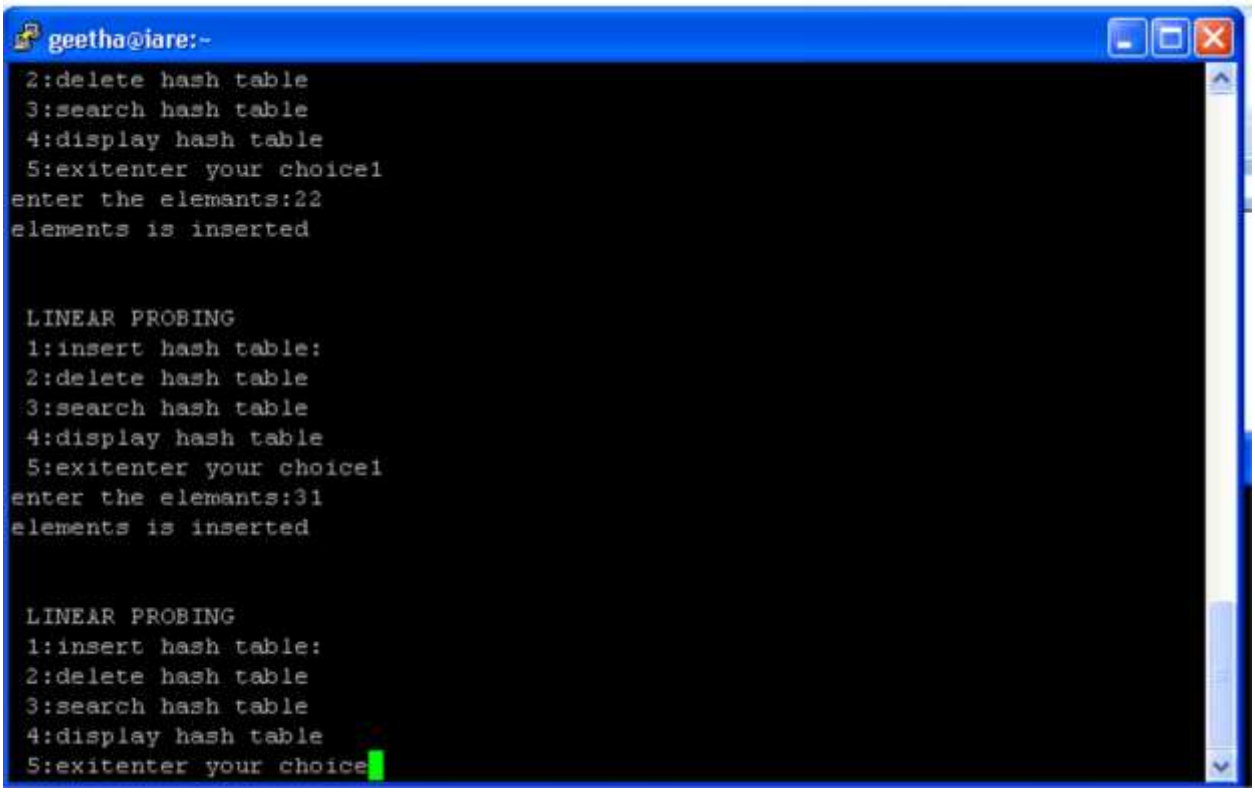
1. Use quadratic probing to fill the Hash table of size 11. Data elements are 23,0,52,61,78,33,100,8,90,10,14.
2. Analyze input (371, 323, 173, 199, 344, 679, 989) and hash function h(x)=x mod 10, Show the result Separate Chaining, linear probing

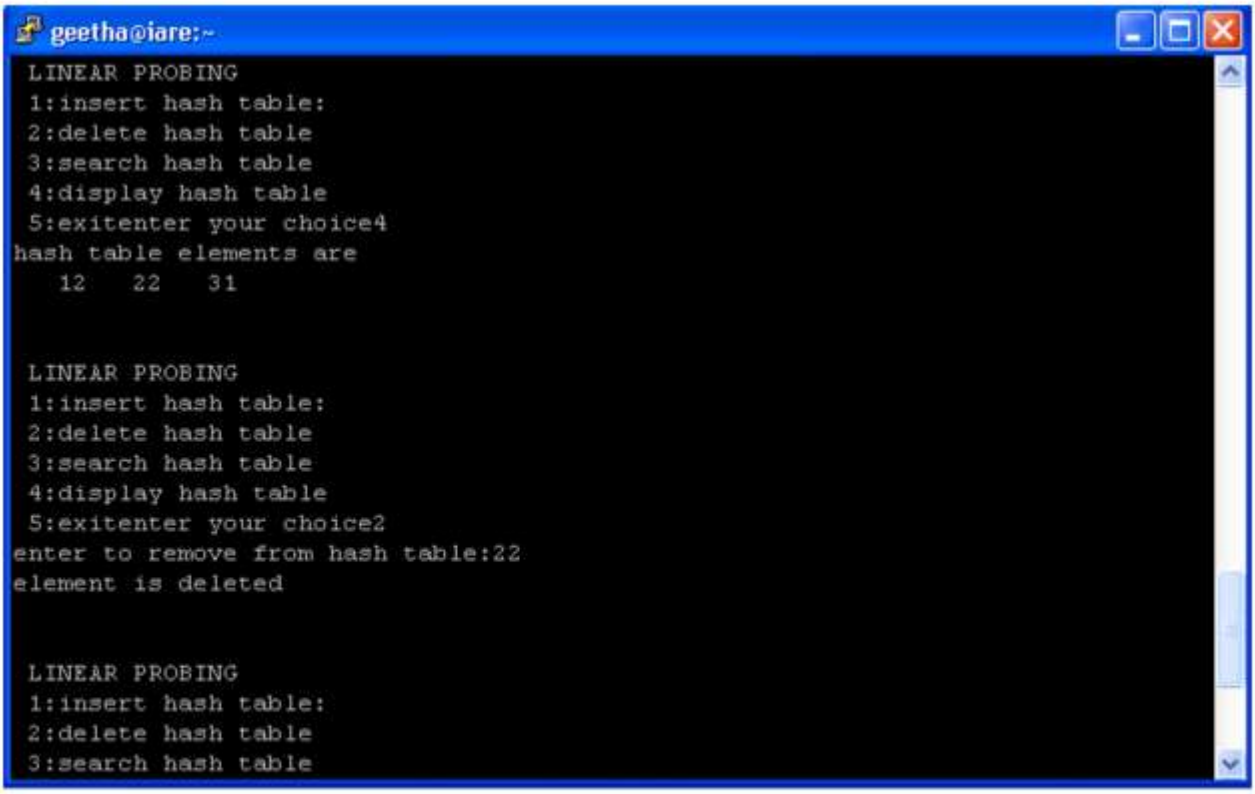
10.8 POST LAB QUESTIONS

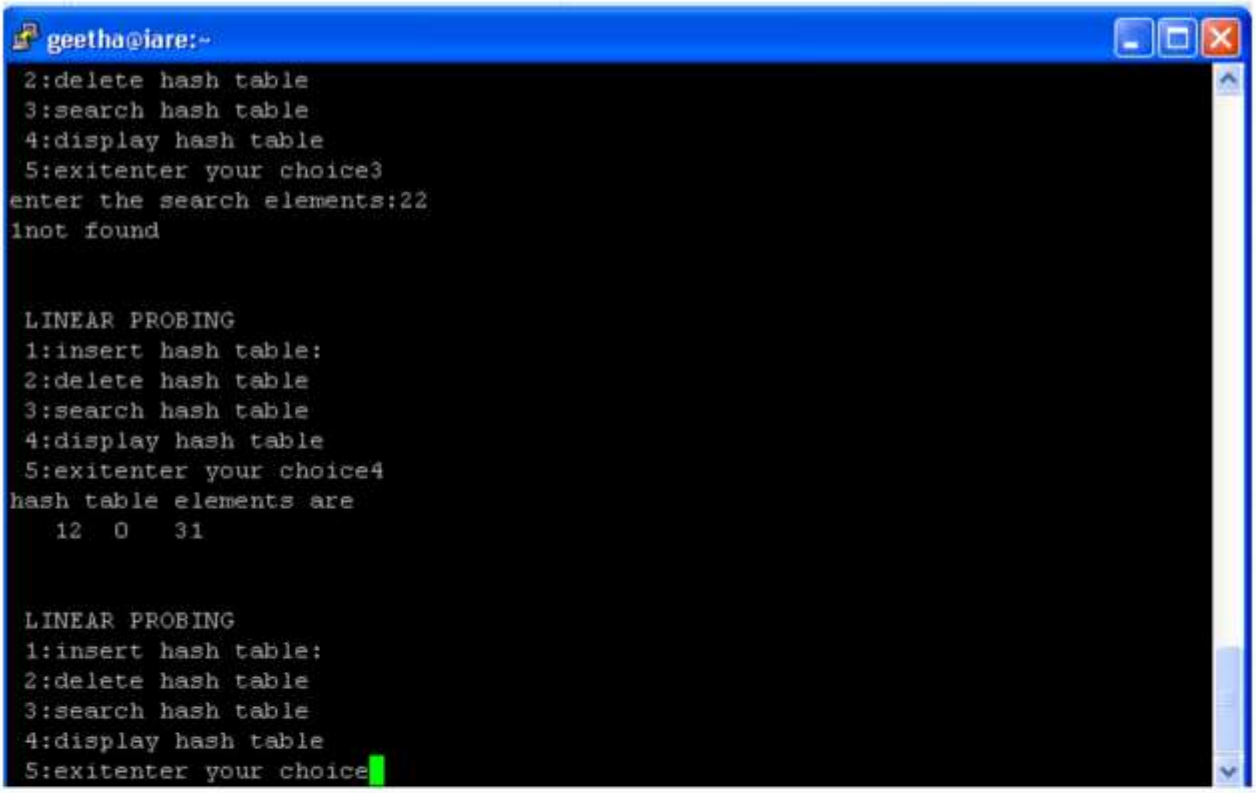
1. List the application of hashing
2. Explain the condition to insert element into hash table
3. What is double hashing

10.9 INPUT AND OUTPUT






geetha@iare:~
LINEAR PROBING
1:insert hash table:
2:delete hash table
3:search hash table
4:display hash table
5:exitenter your choice4
hash table elements are
12 22 31
LINEAR PROBING
1:insert hash table:
2:delete hash table
3:search hash table
4:display hash table
5:exitenter your choice2
enter to remove from hash table:22
element is deleted
LINEAR PROBING
1:insert hash table:
2:delete hash table
3:search hash table


geetha@iare:~
2:delete hash table
3:search hash table
4:display hash table
5:exitenter your choice3
enter the search elements:22
1not found
LINEAR PROBING
1:insert hash table:
2:delete hash table
3:search hash table
4:display hash table
5:exitenter your choice4
hash table elements are
12 0 31
LINEAR PROBING
1:insert hash table:
2:delete hash table
3:search hash table
4:display hash table
5:exitenter your choice

EXPERIMENT 11

11.1 OBJECTIVE
To implementing Knuth-Morris- Pratt pattern matching algorithm

11.2 RESOURCE:
Turbo C

11.3 PROGRAM LOGIC
1. Read the pattern and text
2. Compute failure function
3. Apply the KMP algorithm

11.4 PROCEDURE
Go to debug -> run or press CTRL + F9 to run the program

11.5 SOURCE CODE
Knuth-Morris- Pratt pattern matching algorithm

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

void computefailure(char *pat, int M, int *f);

void KMPSearch(char *pat, char *txt) //to implement kmp search
{
        int M = strlen(pat);//length of pattern
        int N = strlen(txt);//length of text

       int *f = (int *)malloc(sizeof(int)*M); //dynamic allocation
       int j = 0; // index for pat[]

       computefailure(pat, M, f);

        int i = 0;  // index for txt[]
       while(i < N)
      {
        if(pat[j] == txt[i])
        {
          j++;
          i++;
       }
         if (j == M)
       {
         printf("Found pattern at index %d \n", i-j);
         j = f[j-1];
      }
          else if(pat[j] != txt[i])
     {
             if(j != 0)
       j = f[j-1];
       else
        i = i+1;
     }
```

```
    }
    free(f); }

void computefailure (char *pat, int M, int *f) //compute the failure function
{
        int len = 0;
        int i;

        f[0] = 0; // f[0] is always 0
        i = 1;

        while(i < M)
        {
                if(pat[i] == pat[len])
                {
                  len++;
                  f[i] = len;
                  i++;
                }
                else // (pat[i] != pat[len])
                {
                  if( len != 0 )
                  {
                       len = f[len-1];
                  }
                  else   // if (len == 0)
                  {
                    f[i] = 0;
                    i++;
                  }
              }
          }
     }

int main()
{
    int match;
      printf("\nEnter the Text: ");
      gets(text); //reading the text
      printf("\nEnter the Pattern: ");
      gets(pattern); //reading the pattern
      m=strlen(text);
      n=strlen(pattern);
      match=KMPSearch(pat, txt);

      if(match>=0)
      {
                printf("\nMatch found at position %d\n\n",match);
      }
      else
      {
         printf("\nNo Match found!!!\n\n");
      }
      return 0;
}
```

11.6 PRE LAB QUESTIONS

1. what is pattern matching
2. List the various pattern matching algorithms
3. What is failure function

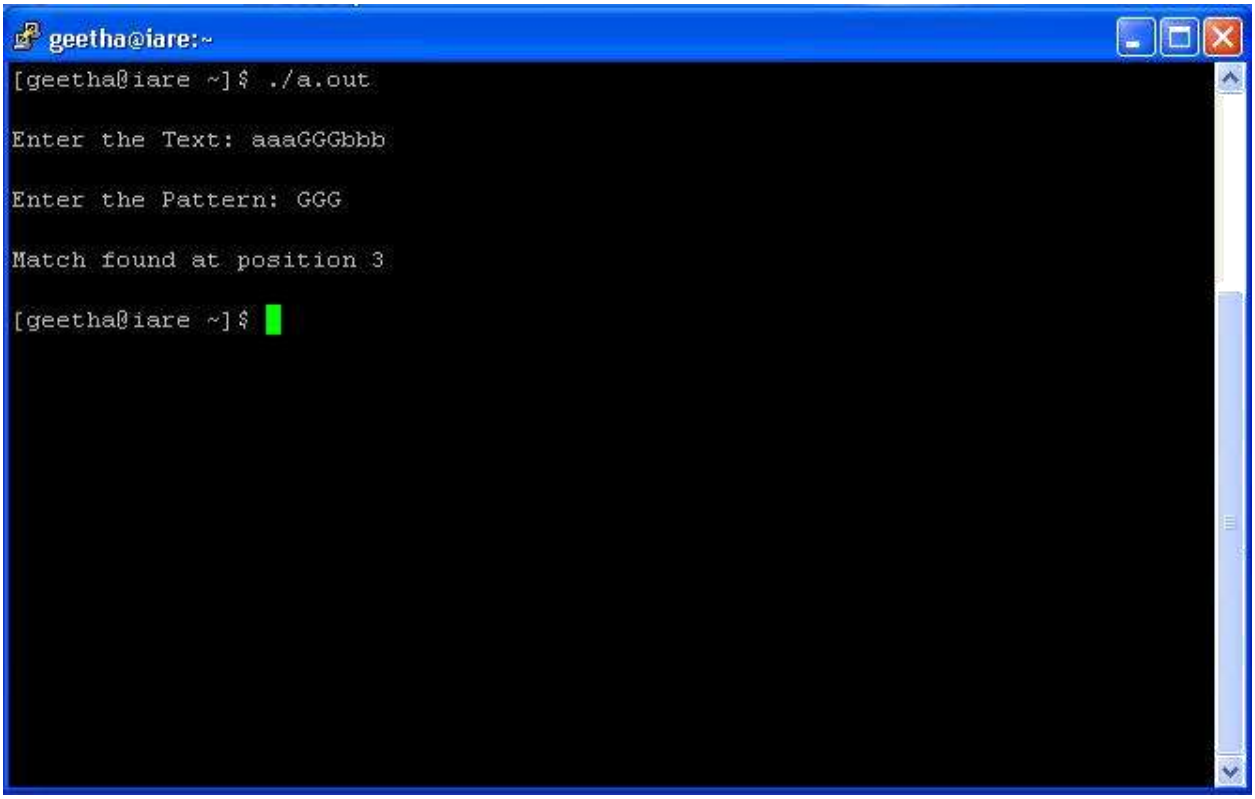
11.7 LAB ASSIGNMENT

1. Apply KMP algorithm on pattern “abacab” and text “abacaabaccabacabaabb”
2. Apply KMP algorithm on pattern “abaa” and text “abbbaababaab”

11.8 POST LAB QUESTIONS

1. Which pattern matching algorithm is better
2. Write the significance of Knuth-Morris- Pratt pattern matching algorithm
3. List the applications of pattern matching algorithms

11. 9 INPUT AND OUTPUT



EXPERIMENT 12

A: Write C programs for implementing the following graph traversal algorithm for Depth first traversal

12.1 OBJECTIVE

To traverse graph

12.2 RESOURCE:

Turbo C

12.3 PROGRAM LOGIC

1. Take the graph as a input
2. Start at some vertex and traverse it using DFS
3. Apply the above procedure for all nodes

12.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

12.5 SOURCE CODE:

DFS

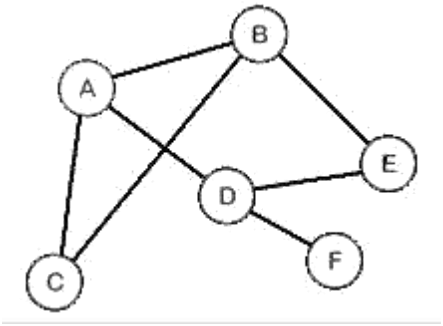
```
#include <stdio.h>
void dfs(int);
int g[10][10],visited[10],n,vertex[10];
void main()
{
   int i,j;
   printf("enter number of vertices:");
    scanf("%d",&n);
    printf("enter the val of vertex:");
    for(i=0;i<n;i++)
            scanf("%d",&vertex[i]);
  printf("\n enter adjecency matrix of the graph:");
  for(i=0;i<n;i++)
             for(j=0;j<n;j++)
            scanf("%d",&g[i][j]);
 for(i=0;i<n;i++)
  visited[i]=0;
   dfs(0);
}
void dfs(int i)
{
  int j;
   printf("%d",vertex[i]);
visited[i]=1;       for(j=0;j<n;j++)
if(!visited[j]&&g[i][j]==1)
  dfs(j);
}
```

12.6 PRE LAB QUESTIONS

1. What is graph
2. List various way of representations of graph
3. How many graph traversal algorithms are there

12.7 LAB ASSIGNMENT

1. Find DFS traversal of the following graph

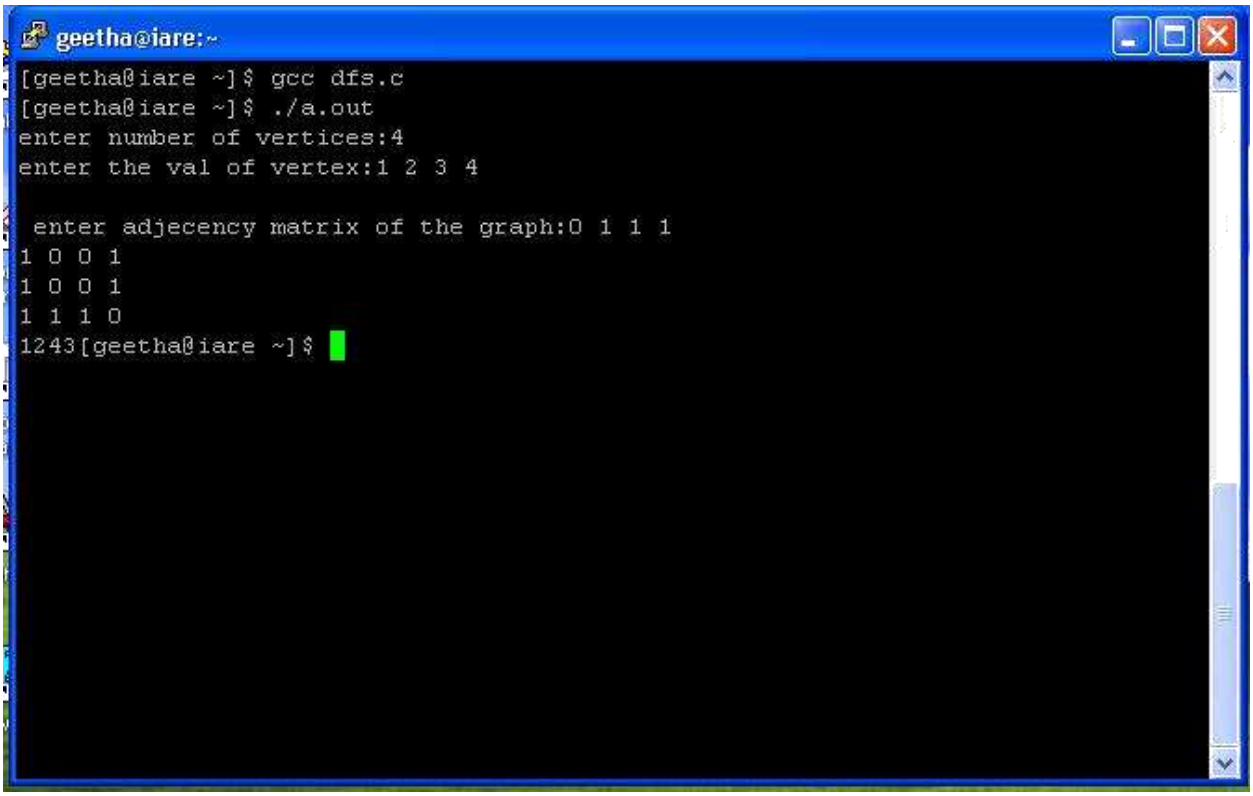


2. Deduce the time complexity of DFS algorithm

12.8 POST LAB QUESTIONS

1. Applications of graph traversals
2. Define minimum spanning tree
3. What is the time complexity of DFS

12. 9 INPUT AND OUTPUT


geetha@iare:~
[geetha@iare ~]$ gcc dfs.c
[geetha@iare ~]$ ./a.out
enter number of vertices:4
enter the val of vertex:1 2 3 4
enter adjecency matrix of the graph:0 1 1 1
1 0 0 1
1 0 0 1
1 1 1 0
1243[geetha@iare ~]$

CONTENT BEYOND SYLLABI

EXPERIMENT 1

1.1 OBJECTIVE

*Write a C Program to check whether two given lists are containing the same data.

1.2 SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>
 struct node
{
   int num;
   struct node *next;
};

void feedmember(struct node **);
int compare (struct node *, struct node *);
void release(struct node **);

int main()  {
   struct node *p = NULL;
   struct node *q = NULL;
   int result;

   printf("Enter data into first list\n");
   feedmember(&p);
   printf("Enter data into second list\n");
   feedmember(&q);
   result = compare(p, q);
   if (result == 1)      {
      printf("The 2 list are equal.\n");
   }
   else    {
      printf("The 2 lists are unequal.\n");
   }
   release (&p);
   release (&q);

   return 0;
}

int compare (struct node *p, struct node *q)   {
   while (p != NULL && q != NULL)
   {
      if (p->num != q-> num)          {
         return 0;
      }
      else
      {
         p = p->next;
         q = q->next;
      }
   }
   if (p != NULL || q != NULL)       {
      return 0;
   }
```

```
   else    {
      return 1;
   }
}

void feedmember (struct node **head)
{
   int c, ch;
   struct node *temp;

   do      {
      printf("Enter number: ");
      scanf("%d", &c);
      temp = (struct node *)malloc(sizeof(struct node));
      temp->num = c;
      temp->next = *head;
      *head = temp;
      printf("Do you wish to continue [1/0]: ");
      scanf("%d", &ch);
   }while (ch != 0);
   printf("\n");
}

void release (struct node **head)  {
   struct node *temp = *head;

   while ((*head) != NULL)
   {
      (*head) = (*head)->next;
      free(temp);
      temp = *head;
   }
}
```

INPUT/ OUTPUT

```
Enter data into first list
Enter number: 12
Do you wish to continue [1/0]: 1
Enter number: 3
Do you wish to continue [1/0]: 1
Enter number: 28
Do you wish to continue [1/0]: 1
Enter number: 9
Do you wish to continue [1/0]: 0

Enter data into second list
Enter number: 12
Do you wish to continue [1/0]: 1
Enter number: 3
Do you wish to continue [1/0]: 1
Enter number: 28
Do you wish to continue [1/0]: 1
Enter number: 9
Do you wish to continue [1/0]: 0

The 2 list are equal.
```

EXPERIMENT 2

2.1 OBJECTIVE

*Write a C program to find the largest element in a given doubly linked list.

2.2 SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
   int num;
   struct node *next;
   struct node *prev;
};

void create(struct node **);
int max(struct node *);
void release(struct node **);

int main()
{
   struct node *p = NULL;
   int n;

   printf("Enter data into the list\n");
   create(&p);
   n = max(p);
   printf("The maximum number entered in the list is %d.\n",
   n); release (&p);

   return 0;
}

int max(struct node *head)
{
   struct node *max, *q;

   q = max = head;
   while (q != NULL)
   {
      if (q->num > max->num)
      {
         max = q;
      }
      q = q->next;
   }

   return (max->num);
}

void create(struct node **head)
{
   int c, ch;
   struct node *temp, *rear;
```

```
    do
    {
        printf("Enter number: ");
        scanf("%d", &c);
        temp = (struct node *)malloc(sizeof(struct node));
        temp->num = c;
        temp->next = NULL;
        temp->prev = NULL;
        if (*head == NULL)
        {
            *head = temp;
        }
        else
        {
            rear->next = temp;
            temp->prev = rear;
        }
        rear = temp;
        printf("Do you wish to continue [1/0]: ");
        scanf("%d", &ch);
    } while (ch != 0);
    printf("\n");
}

void release(struct node **head)
{
    struct node *temp = *head;
    *head = (*head)->next;
    while ((*head) != NULL)
    {
        free(temp);
        temp = *head;
        (*head) = (*head)->next;
    }
}
```

INPUT/ OUTPUT:

```
Enter data into the list
Enter number: 12
Do you wish to continue [1/0]: 1
Enter number: 7
Do you wish to continue [1/0]: 1
Enter number: 23
Do you wish to continue [1/0]: 1
Enter number: 4
Do you wish to continue [1/0]: 1
Enter number: 1
Do you wish to continue [1/0]: 1
Enter number: 16
Do you wish to continue [1/0]: 0

The maximum number entered in the list is 23.
```

EXPERIMENT 3

3.1 OBJECTIVE

*Write a C program to reverse the elements in the stack using recursion.

3.2 SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
   int a;
   struct node *next;
};

void generate(struct node **);
void display(struct node *);
void stack_reverse(struct node **, struct node **);
void delete(struct node **);

int main()
{
   struct node *head = NULL;

   generate(&head);
   printf("\nThe sequence of contents in stack\n");
   display(head);
   printf("\nInversing the contents of the stack\n");
   if (head != NULL)
   {
      stack_reverse(&head, &(head->next));
   }
   printf("\nThe contents in stack after reversal\n");
   display(head);
   delete(&head);

   return 0;
}

void stack_reverse(struct node **head, struct node **head_next)
{
   struct node *temp;

   if (*head_next != NULL)
   {
       temp = (*head_next)->next;
      (*head_next)->next = (*head);
      *head = *head_next;
      *head_next = temp;
      stack_reverse(head, head_next);
   }
}

void display(struct node *head)
{
```

```
    if (head != NULL)
    {
        printf("%d ", head->a);
        display(head->next);
    }
}

void generate(struct node **head)
{
    int num, i;
    struct node *temp;

    printf("Enter length of list: ");
    scanf("%d", &num);
    for (i = num; i > 0; i--)
    {
        temp = (struct node *)malloc(sizeof(struct node));
        temp->a = i;
        if (*head == NULL)
        {
            *head = temp;
            (*head)->next = NULL;
        }
        else
        {
            temp->next = *head;
            *head = temp;
        }
    }
}

void delete(struct node **head)
{
    struct node *temp;
    while (*head != NULL)
    {
        temp = *head;
        *head = (*head)->next;
        free(temp);
    }
}
```

INPUT/OUTPUT:

```
Enter length of list: 10

The sequence of contents in stack
12345678910

Reversing the contents of the stack

The contents in stack after reversal
10987654321
```

EXPERIMENT 4

4.1 OBJECTIVE

*Write a C program to implement stack using linked list.

4.2 SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
   int info;
   struct node *ptr;
}*top,*top1,*temp;

int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();

int count = 0;

void main()
{
   int no, ch, e;

   printf("\n 1 - Push");
   printf("\n 2 - Pop");
   printf("\n 3 - Top");
   printf("\n 4 - Empty");
   printf("\n 5 - Exit");
   printf("\n 6 - Dipslay");
   printf("\n 7 - Stack Count");
   printf("\n 8 - Destroy stack");

   create();

   while (1)
   {
      printf("\n Enter choice : ");
      scanf("%d", &ch);

      switch (ch)
      {
      case 1:
         printf("Enter data : ");
         scanf("%d", &no);
         push(no);
         break;
      case 2:
         pop();
         break;
```

```
        case 3:
            if (top == NULL)
                printf("No elements in stack");
            else
            {
                e = topelement();
                printf("\n Top element : %d", e);
            }
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            stack_count();
            break;
        case 8:
            destroy();
            break;
        default :
            printf(" Wrong choice, Please enter correct choice ");
            break;
        }
    }
}

/* Create empty stack */
void create()
{
    top = NULL;
}

/* Count stack elements */
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
```

```
    count++;
}

/* Display stack elements */
void display()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }

    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
 }

/* Pop Operation on stack */
void pop()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
    count--;
}

/* Return top element */
int topelement()
{
    return(top->info);
}

/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}

/* Destroy entire stack */
void destroy()
{
    top1 = top;
```

```
    while (top1 != NULL)
    {
        top1 = top->ptr;
        free(top);
        top = top1;
        top1 = top1->ptr;
    }
    free(top1);
    top = NULL;

    printf("\n All stack elements destroyed");
    count = 0;
}
```

INPUT/OUTPUT:

```
1 - Push
2 - Pop
3 - Top
4 - Empty
5 - Exit
6 - Dipslay
7 - Stack Count
8 - Destroy stack
Enter choice : 1
Enter data : 56

Enter choice : 1
Enter data : 80

Enter choice : 2

Popped value : 80
Enter choice : 3

Top element : 56
Enter choice : 1
Enter data : 78

Enter choice : 1
Enter data : 90

Enter choice : 6
90 78 56
Enter choice : 7

No. of elements in stack : 3
Enter choice : 8

All stack elements destroyed
Enter choice : 4

Stack is empty
Enter choice : 5
```

EXPERIMENT 5

5.1 OBJECTIVE

*Write a C program to count the number of nodes in the binary search tree.

5.2 SOURCE CODE

```
#include < stdio.h >
#include < conio.h >
#include < alloc.h >
#define new_node (struct node*)malloc(sizeof (struct node))

struct node
{
        int data;
        struct node *lchild;
        struct node *rchild;
};

struct node *create_bin_rec();
void print_bin_pre_rec(struct node *t);
void cnt_nodes(struct node *t, int *l, int *nl);

void main()
{
        struct node *root;
        int leaf,nonleaf;
        clrscr();
        printf("\nCreate a binary tree \n");
        root = create_bin_rec();
        printf("\n Binary tree is ");
        print_bin_pre_rec(root);
        leaf = nonleaf = 0;
        cnt_nodes(root,&leaf,&nonleaf);
        printf("\n Total  no. of leaf nodes are : %d ",leaf);
        printf("\n Total  no. of nonleaf nodes are : %d ",nonleaf);
        printf("\n Total  no. of nodes are : %d ", (leaf+nonleaf));

} // main

struct node *create_bin_rec()
{
        int data;
        struct node *t;
        printf("\nData ( -1 to exit ) : ");
        scanf("%d",&data);
        if( data == -1)
                return(NULL);
        else
        {
                t = new_node;
                t->data = data;
                t->lchild =create_bin_rec();
                t->rchild =create_bin_rec();
                return(t);
        } //else
```

```
} // create

void print_bin_pre_rec(struct node *t)
{
        if( t != NULL)
        {
                printf("%4d",t->data);
                print_bin_pre_rec(t->lchild);
                print_bin_pre_rec(t->rchild);
        } // if
} // print bin pre rec

void cnt_nodes(struct node *t, int *l, int *nl)
{
        if( t != NULL)
        {
                if( t->lchild == NULL && t->rchild == NULL)
                        (*l)++;
                else
                        (*nl)++;
                        cnt_nodes(t->lchild,l,nl);
                        cnt_nodes(t->rchild,l,nl);
        } // if
} // cnt nodes
```

INPUT OUTPUT

```
Create binary tree
Data (-1 to exit) 10
Data (-1 to exit) 20
Data (-1 to exit)  -1

Binary tree is 10 20
Total no. of leaf nodes are 1
Total no. of non leaf nodes are 1
Total no. of nodes are 2
```

EXPERIMENT 6

6.1 OBJECTIVE

*Write a C program to sort an array of integers in ascending order using radix sort.

6.2 SOURCE CODE

```
#include <stdio.h>
int min = 0, count = 0, array[100] = {0}, array1[100] = {0};

void main()
{
   int k, i, j, temp, t, n;

   printf("Enter size of array :");
   scanf("%d", &count);
   printf("Enter elements into array :");
   for (i = 0; i < count; i++)
   {
      scanf("%d", &array[i]);
      array1[i] = array[i];
   }
   for (k = 0; k < 3; k++)
   {
      for (i = 0; i < count; i++)
      {
         min = array[i] % 10;        /* To find minimum lsd */
         t = i;
         for (j = i + 1; j < count; j++)
         {
            if (min > (array[j] % 10))
            {
               min = array[j] % 10;
               t = j;
            }
         }
         temp = array1[t];
         array1[t] = array1[i];
         array1[i] = temp;
         temp = array[t];
         array[t] = array[i];
         array[i] = temp;

      }
      for (j = 0; j < count; j++)       /*to find MSB */
         array[j] = array[j] / 10;
   }
   printf("Sorted Array (lSdradix sort) : ");
   for (i = 0; i < count; i++)
      printf("%d ", array1[i]);
}
```

INPUT/ OUTPUT:

```
/* Average Case */
Enter size of array :7
```

```
Enter elements into array :170
45
90
75
802
24
2
Sorted Array (ladradix sort) : 2 24 45 75 90 170 802

/*Best case */
Enter size of array :7
Enter elements into array :22
64
121
78
159
206
348
Sorted Array (ladradix sort) : 22 64 78 159 121 206 348

    /* Worst case */
Enter size of array :7
Enter elements into array :985
27
64
129
345
325
091
Sorted Array (ladradix sort) : 27 64 91 129 325 345 985
```

EXPERIMENT 7

7.1 OBJECTIVE

*Write a C program to sort a given list of strings.

7.2 SOURCE CODE

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main() {
char *str[5], *temp;
 int i, j, n;
 printf("\nHow many names do you want to have?");
 scanf("%d", &n);
 for (i = 0; i < n; i++) {
      printf("\nEnter the name %d: ", i);
      flushall();
      gets(str[i]);
}
for (i = 0; i < n; i++) {
     for (j = 0; j < n - 1; j++) {
       if (strcmp(str[j], str[j + 1]) > 0) {
           strcpy(temp, str[j]);
           strcpy(str[j], str[j + 1]);
           strcpy(str[j + 1], temp);
} } }
printf("\nSorted List : ");
for (i = 0; i < n; i++)
      puts(str[i]);
  return (0);
}
```

INPUT /OUTPUT:

```
Enter any strings :
pri
pra
pru
pry
prn


Strings in order are :
pra
pri
prn
pru
```

pry

LAB MANUAL
FOR
DATA STRUCTURE USING C LAB

WCTM

WORLD COLLEGE OF TECHNOLOGY & MANAGEMENT
WCTM
Promoting Excellence

PROGRAM NO.1

Aim: - To search an element in the array using Linear Search.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[10],i,item,flag=0;
        clrscr();
        printf("Enter the data in the array");
        for(i=0;i<10;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("Enter the element to be searched");
        scanf("%d",&item);
        for(i=0;i<10;i++)
        {
                if(item==a[i])
                {
                        flag=1;
                        break;
                }
        }
        if(flag==0)
        printf("Element Not Found");
        else
        printf("Element Found at Position =%d",i);
getch();
}
```

WCTM

PROGRAM NO.2

Aim: - To search an element in the 2-dimensional array using Linear Search.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[3][3],i,j,item,flag=0;
        clrscr();
        printf("Enter the data in the array");
        for(i=0;i<3;i++)
        {
        for(j=0;j<3;j++)
                {
                scanf("%d",&a[i][j]);
                }
        }
        printf("Enter the element to be searched");
        scanf("%d",&item);
        for(i=0;i<3;i++)
        {
        for(j=0;j<3;j++)
        {
                if(item==a[i][j])
                {
                        flag=1;
                       printf("Element found at position =%d,%d",i,j);
                }
        }
        }
        if(flag==0)
        printf("Element Not Found");

getch();
```

```
}
```

PROGRAM NO.3

Aim: - To merge two sorted array into one sorted array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[10],b[10],c[20],i,j,k,n,m,t;
        clrscr();
        printf("Enter size of Array A\n");
        scanf("%d",&n);
        printf("Enter the data in Array A\n");
        for(i=0;i<n;i++)
        {
        scanf("%d",&a[i]);
        }
        printf("Enter size of Array B\n");
        scanf("%d",&m);
        printf("Enter the data in Array B\n");
        for(j=0;j<m;j++)
        {
        scanf("%d",&b[j]);
        }
        i=j=k=0;
        while(i<n&&j<m)
        {
                if(a[i]<b[j])
                c[k++]=a[i++];
                else
                if(a[i]>=b[j])
                c[k++]=b[j++];
        }
```

WCTM

```
        if(i<n)
        {
                for(t=0;t<n;t++)
                c[k++]=a[i++];
        }
        else
        {
                for(t=0;t<m;t++)
                  c[k++]=b[j++];
        }
          printf("\n");
          for(k=0;k<(m+n);k++)
          printf("\n %d ",c[k]);
  getch();
}
```

WCTM

PROGRAM NO.4

Aim: - To perform the following operation in Matrix

1. Addition 2. Subtraction 3. Multiplication 4. Transpose

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[3][3],b[3][3],c[3][3],d[3][3],i,j,k;
        clrscr();
        printf("Enter the data in Matrix A");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                scanf("%d",&a[i][j]);
                }
        }
        printf("Enter the data in Martix B");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                scanf("%d",&b[i][j]);
                }
        }
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
```

WCTM

```
            {
            c[i][j]=a[i][j]+b[i][j];
            }
}
printf("Addition of two Matrix A and B is\n");
for(i=0;i<3;i++)
{
            for(j=0;j<3;j++)
            {
            printf("%d\t",c[i][j]);
            }
            printf("\n");
}
            for(i=0;i<3;i++)
{
            for(j=0;j<3;j++)
            {
            c[i][j]=a[i][j]-b[i][j];
            }
}
printf("Subtraction of two Matrix A and B is\n");
for(i=0;i<3;i++)
{
            for(j=0;j<3;j++)
            {
            printf("%d\t",c[i][j]);
            }
            printf("\n");
}
printf("Transpose of Matrix C is\n");
for(i=0;i<3;i++)
{
            for(j=0;j<3;j++)
            {
            d[j][i]=c[i][j];
            }
}
 for(i=0;i<3;i++)
{
```

WCTM

```
                for(j=0;j<3;j++)
                {
                printf("%d\t",d[i][j]);
                }
                printf("\n");
        }
                printf("Multiplication of Matrix A and B is\n");

                for(i=0;i<3;i++)
                {
                        for(j=0;j<3;j++)
                        {
                        c[i][j]=0;
                        for(k=0;k<3;k++)
                        {
                         c[i][j]=c[i][j]+a[i][k]*b[k][j];
                        }
                        }
                }
                printf("\n");
                for(i=0;i<3;i++)
                {
                for(j=0;j<3;j++)
                {
                printf("%d\t",c[i][j]);
                }
                printf("\n");
                }
getch();
}
```

WCTM

PROGRAM NO.5

Aim: - To perform the swapping of two numbers using call by value and call by reference.

```
#include<stdio.h>
#include<conio.h>
void swapbyvalue(int,int);
void swapbyref(int*,int*);
void main()
{
        int a,b;
        clrscr();
        printf("Enter the two numbers");
        scanf("%d%d",&a,&b);
        swapbyvalue(a,b);
        swapbyref(&a,&b);
        printf("\nNumber after swapping by Reference\n");
        printf("\na=%d\nb=%d",a,b);
        getch();
}
void swapbyvalue(int x, int y)
{
        int temp;
        temp=x;
        x=y;
```

WCTM

```
        y=temp;
        printf("\nNumbers after swapping by value are\n");
        printf("a=%d",x);
        printf("\nb=%d",y);
}
void swapbyref(int *x,int *y)
{
        int temp;
        temp=*x;
        *x=*y;
        *y=temp;
}
```

PROGRAM NO.6

Aim: - To perform following operation on strings using string functions
1. Addition 2. Copying 3. Reverse 4. Length of String.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
        char a[20],b[20],c[20];
        int l;
        clrscr();
        printf("Enter the First String");
        scanf("%s",&a);
        printf("Enter the Second String");
        scanf("%s",&b);
        strcat(a,b);
        printf("\nConcatenation of String a and b is:%s",a);
        l=strlen(a);
        printf("\nLength of String is %d",l);
        strcpy(c,a);
        printf("\nthe Copied String is %s",c);
        strrev(a);
```

```
        printf("\nreverse of String is %s",a);

        getch();
}
```

WCTM

PROGRAM NO.7 (a)

Aim: - To search an element in the array using Iterative Binary Search.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[20],n,mid,beg,i,end,item,loc=-1;
        clrscr();
        printf("Enter the number of elements to be entered\n");
        scanf("%d",&n);
        printf("Enter the elements in ascending order");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("Enter the element to be searched");
        scanf("%d",&item);
        beg=0;
        end=n-1;
        while(beg<=end)
```

```
        {
                mid=(beg+end)/2;
                if(item==a[mid])
                {
                loc=mid;
                break;
                }
        else if(a[mid]<item)
        beg=mid+1;
        else
        end=mid-1;
        }
        if(loc==-1)
        printf("Element Not Present");
        else
        printf("Element found at =%d",loc);
        getch();
 }
```

PROGRAM NO.7 (b)

Aim: - To search an element in the array using Recursive Binary Search.

```
#include<stdio.h>
#include<conio.h>
void binary(int [],int,int);
void main()
{
        int a[20],i,n,item;
        clrscr();
        printf("Enter the number of items in the array");
        scanf("%d",&n);
        printf("enter the data in array");
        for(i=0;i<n;i++)
        {
         scanf("%d",&a[i]);
        }
        printf("Enter the element to be searched");
```

```
        scanf("%d",&item);

        binary(a,n,item);
getch();
}

void binary(int a[],int n,int item)
{
 int beg,end,mid,loc=-1;
 beg=0;
 end=n-1;
 while(beg<=end)
   {
        mid=(beg+end)/2;
        if(item==a[mid])
        {
                loc=mid;
                break;
        }

else if(item>a[mid])
        beg=mid+1;
        else
        end=mid-1;
   }
   if(loc==-1)
   printf("Element not Found");
   else
   printf("Element Found at position = %d",loc);
}
```

WCTM

WCTM

PROGRAM NO.8

Aim: - To implement Bubble Sort.

```
#include<stdio.h>
#include<conio.h>
void bubble(int [],int);
void main()
{
        int a[20],i,n;
        clrscr();
        printf("Enter the number of items in the array");
        scanf("%d",&n);
        printf("Enter the data in the array");
        for(i=0;i<n;i++)
        {
         scanf("%d",&a[i]);
```

```
        }
        bubble(a,n);
getch();
}
void bubble(int a[],int n)
{
    int i,temp,j,p;
    for(i=1;i<n;i++)
    {
        for(p=0;p<n-i;p++)
           {
                if(a[p]>a[p+1])
                {
                        temp=a[p];
                        a[p]=a[p+1];
                        a[p+1]=temp;
                }
           }
    }
    for(i=0;i<n;i++)
    printf("\n%d",a[i]);
}
```

PROGRAM NO.9

Aim: - To implement Selection Sort.

```
#include<stdio.h>
#include<conio.h>
void select(int [],int);
void bubble(int [],int);
int min(int [],int,int);

void main()
{
        int a[20],i,n;
        clrscr();
        printf("Enter the number of items in the array");
```


WCTM

```
        scanf("%d",&n);
        printf("Enter the data in the array");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        bubble(a,n);
        select(a,n);
        getch();
}
void bubble(int a[],int n)
{
    int i,temp,p;
    for(i=1;i<n;i++)
    {
        for(p=0;p<n-i;p++)
           {
                if(a[p]>a[p+1])
                {
                        temp=a[p];
                        a[p]=a[p+1];
                        a[p+1]=temp;
                }
           }
    }
    printf("\nData After Bubble Sort");
    for(i=0;i<n;i++)
    printf("\n%d",a[i]);
}

void select(int a[],int n)
{
        int i,loc,temp;
        loc=0;
        temp=0;
        for(i=0;i<n;i++)
        {
                loc=min(a,i,n);
                temp=a[loc];
```

```
                a[loc]=a[i];
                a[i]=temp;
        }
        printf("\nData After Selection Sort");
        for(i=0;i<n;i++)
        printf("\n%d",a[i]);
}

int min(int a[],int lb,int ub)
{
        int m=lb;
        while(lb<ub)
        {
                if(a[lb]<a[m])
                {
                m=lb;
                }
        lb++;
        }
        return m;
}
```

PROGRAM NO.10

Aim: - To implement Insertion Sort.

```
#include<stdio.h>
#include<conio.h>

void insert(int [],int);
void main()
{
        int a[20],i,n;
```


WCTM

```
        clrscr();
        printf("Enter the number of items in the array");
        scanf("%d",&n);
        printf("Enter the data in the array");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        insert(a,n);
        getch();
}


void insert(int a[],int n)
{
        int i,j,temp;
        for(i=1;i<n;i++)
        {
                temp=a[i];
                for(j=i-1;j>=0;j--)
                {
                        if(a[j]>temp)
                        {
                        a[j+1]=a[j];
                        }
                        else
                        break;
                }
                a[j+1]=temp;
        }
        printf("Data After Insertion Sort");
        for(i=0;i<n;i++)
        printf("\n%d",a[i]);

}
```

WCTM

WCTM

PROGRAM NO.11

Aim: - To implement Quick Sort.

```
#include<stdio.h>
#include<conio.h>

void quicksort(int[],int,int);
int partition(int [],int,int);
```

```
void main()
{
        int a[20],i,n;
        clrscr();
        printf("Enter the size of array");
        scanf("%d",&n);
        printf("Enter the elements in the array");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        quicksort(a,0,n-1);
        for(i=0;i<n;i++)
        printf("\n%d",a[i]);
        getch();
}

void quicksort(int a[],int lb,int ub)
{
        int mid;
        if(lb<ub)
        {
                mid=partition(a,lb,ub);
                quicksort(a,lb,mid-1);
                quicksort(a,mid+1,ub);
        }
}


int partition(int a[],int lb,int ub)
{
        int i,p,q,t;
        p=lb+1;
        q=ub;
        i=a[lb];

        while(q>=p)
```

WCTM

```
        {
                while(a[p]<i)
                p++;
                while(a[q]>i)
                q--;
                if(q>p)
                {
                        t=a[p];
                        a[p]=a[q];
                        a[q]=t;
                }
        }

        t=a[lb];
        a[lb]=a[q];
        a[q]=t;
        return q;
}
```

WCTM

PROGRAM NO.12

Aim: - To implement Merge Sort.

```
#include<stdio.h>
#include<conio.h>
void mergesort(int a[],int,int);
void merge(int [],int,int,int);
void main()
{
        int a[20],i,n;
        clrscr();
        printf("Enter the number of elements");
        scanf("%d",&n);
        printf("Enter the elements");
        for(i=0;i<n;i++)
        {
        scanf("%d",&a[i]);
        }
        mergesort(a,0,n-1);
        printf("Data After Merge Sort");
        for(i=0;i<n;i++)
        printf("\n%d",a[i]);
        getch();
}
void mergesort(int a[],int lb,int ub)
{
        int mid;
        if(lb<ub)
        {
                mid=(lb+ub)/2;
                mergesort(a,lb,mid);
                mergesort(a,mid+1,ub);
                merge(a,lb,mid+1,ub);
        }
}


void merge(int a[],int lb,int mid,int ub)
{
        int k,p1,p2,p3,b[20];
        p1=lb;
        p3=lb;
```

```
        p2=mid;
        while((p1<mid)&&(p2<=ub))
        {
                if(a[p1]<=a[p2])

                b[p3++]=a[p1++];

                else

                b[p3++]=a[p2++];

        }
        while(p1<mid)
        {
                b[p3++]=a[p1++];
        }
        while(p2<=ub)
        {
                b[p3++]=a[p2++];
        }
        for(k=lb;k<p3;k++)
        {
        a[k]=b[k];
        }
}
```

WCTM

PROGRAM NO.13

Aim: - To implement Stack using array.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>

void push();
void pop();
void display();

int top;
int a[5];

void main()
{
        int choice;
        char ch;
        top=-1;
        clrscr();
        do
        {
        printf("\n\t 1. PUSH");
        printf("\n\t 2. POP");
        printf("\n\t 3. DISPLAY");
        printf("\n\t 4. EXIT");
        printf("\nEnter your choice");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
        push();
        break;
        case 2:
        pop();
        break;

        case 3:
        display();
```

WCTM

```
        break;
        case 4:
        exit(0);
        default:
        printf("\nBAD CHOICE");
        }
        printf("\ndo you want to continue y/n");
        ch=getche();
        }
        while(ch=='y');
}

void push()
{
        int item;
        if(top==4)
        printf("STACK IS FULL");
        else
        {
                printf("Enter the item to be inserted");
                scanf("%d",&item);
                top=top+1;
                a[top]=item;
                //top=tope;
        }
}

void pop()
{
        int item;
        if(top==-1)
        printf("STACK IS EMPTY");
        else
        {
                item=a[top];
                top=top-1;
                printf("%d is deleted",item);
                //top=tope;
        }
```

WCTM

```
}


void display()
{
        int i;
        for(i=top;i>=0;i--)
        printf("\n%d",a[i]);
}
```

WCTM

PROGRAM NO.14

Aim: - To implement Queue using array.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>

void insert();
void delet();
void display();
int front,rear;
int q[5];

void main()
{
        int choice;
        char ch;
        front=-1;
        rear=-1;
        clrscr();
        do
        {
        printf("\n\t 1. INSERT");
        printf("\n\t 2. DELETE");
        printf("\n\t 3. DISPLAY");
        printf("\n\t 4. EXIT");
        printf("\nEnter your choice");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
        insert();
        break;
        case 2:
        delet();
        break;
```

WCTM

```
        case 3:
        display();
        break;
        case 4:
        exit(0);
        default:
        printf("\nBAD CHOICE");
        }
        printf("\ndo you want to continue y/n");
        ch=getche();
        }
        while(ch=='y'||'Y');
}

void insert()
{
        int item;
        if(((front==1)&&(rear==5))||(front==rear+1))
        {
        printf("QUEUE IS FULL");
        }
        else
        {
                printf("Enter the element");
                scanf("%d",&item);
                if(front==-1)
                {
                        front=1;
                        rear=1;
                }
                else if(rear==5)
                {
                        rear=0;
                }
                else
                {
                        rear=rear+1;
                }
                q[rear]=item;
```

WCTM

```
        }
}

void delet()
{
        int item;
        if(front==-1)
        {
         printf("QUEUE IS EMPTY");
        }
        else
        {
                item=q[front];
                if(front==rear)
                {
                        front=-1;
                        rear=-1;
                }
                else if(front==5)
                {
                        front=0;
                }
                else
                front=front+1;
                printf("%d is deleted",item);
        }
}

void display()
{
        int i;
        if(front==-1)
        printf("QUEUE IS EMPTY");
        else
        {
        for(i=front;i<=rear;i++)
        {
        printf("\n%d",q[i]);
        }}
```

```
}
```

PROGRAM NO.15

Aim: - To implement Linked List.

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>

struct node
{
        int info;
        struct node *next;
};
struct node *start=NULL;

void ins();
void ins_at_beg
();
void ins_at_mid();
void  ins_at_end();
void del();
void  del_at_beg();
void del_at_mid();
void  del_at_end();
void display();
int count();

void main()
{
        int ch=0,i=0,cnt;
        clrscr();
        while(1)
        {
                printf("***********menu***********");
                printf("\n1.insert");

                printf("\n2.delete");
```

```
                printf("\n3.display");
                printf("\n4.count");
                printf("\n5.exit");
                printf ("\nenter your choice : ");
                scanf("%d",&ch);


                switch(ch)
                {
                        case 1:ins();
                        break;
                        case 2:del();
                        break;
                        case 3:display();
                        break;
                        case 4:cnt=count();
                                printf("\n the no of nodes : %d\n",cnt);
                        break;
                        case 5:exit(1);

                }
        }
}

void ins()
{
        int j=0,ch1=0;
        printf("\nenter your choice");
        printf("\n1.insert at the beggning");
        printf("\n2.insert at the middle");
        printf("\n3.insert at the end");
        scanf ("%d",&ch1);
        switch(ch1)
        {
                case 1:ins_at_beg();
                break;
                case 2:ins_at_mid();
                break;
                case 3:ins_at_end();


        }
}
void ins_at_beg()
{
```

```
        int info;
        struct node *t=(struct node *)malloc(sizeof(struct node));
        printf("\nenter information to be inserted in the beggning");
        scanf("%d",&info);
        t->info=info;
        t->next=start;
        start=t;
}
void ins_at_mid()
{
        int inform,x,i;
        struct node *t=(struct node *)malloc(sizeof(struct node));
        struct node *p=start;
        printf("\nenter the location after which new node to be added");
        scanf("%d",&x);
        for(i=1;i<x;i++)
                p=p->next;
        printf("\nenter information of the new node");
        scanf("%d",&inform);
        t->info=inform;
        t->next=p->next;
        p->next=t;
}

void ins_at_end()
{
        int inform1;
        struct node *t=(struct node *)malloc(sizeof(struct node));
        struct node *p=start;

        printf("\nenter information to be added");
        scanf("%d",&inform1);
        t->info=inform1;
        while(p->next!=NULL)
                p=p->next;

        p->next=t;
        t->next=NULL;
}

void del()
{
        int k=0,ch2=0;
        printf("\nenter your choice");
        printf("\n1.delete at the beggning");
```

```
        printf("\n2.delete at the middle");
        printf("\n3.delete at the end");
        scanf ("%d",&ch2);
        switch(ch2)
        {
                case 1:del_at_beg();
                break;
                case 2:del_at_mid();
                break;
                case 3:del_at_end();
                break;
        }

}

void del_at_beg()
{
        struct node *t=start;
        start=start->next;
        free(t);
}

void del_at_mid()

{
        int n;
        struct node *cur=start;
        struct node *pre=start;
        printf("\nenter information to be deleted");
        scanf("%d",&n);
        while(cur->info!=n)
        {
                pre=cur;
                cur=cur->next;
        }
        pre->next=cur->next;
        free(cur);
}

void del_at_end()
{
        struct node *cur=start;
        struct node *pre=start;
        while(cur->next!=NULL)
        {
```

```
                pre=cur;
                cur=cur->next;
        }
        pre->next=NULL;
        free(cur);
}
void display()
{

        struct node *p=start;
        printf("\n\n**********LINK LIST**********\n\n");
        while(p!=NULL)
        {
                printf("%d\n",p->info);
                p=p->next;
        }


}
int count()
{
```

WCTM

```
            int c=0;
            struct node
            *q=start;
            while(q!=NULL)
            {
                    q=q->next;

                    c=c+1;
            }
    return c;
  }
// Problem Statement - insert element in to an array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
void main()
{
int array[200], index = -1;
int i;
while(1)
{
printf("\n1. Insert a new element in to array\n2. update a element\n3. display array
\n4. exit()\nEnter Your choice - ");
scanf(" %d", &i);
switch(i)
{
case 1: if(index < 200)
{
index++;
printf("\nEnter value for new element - ");
scanf(" %d", &array[index]);
} else
printf("\narray is full !");
break;
case 2: printf("\nEnter value of index to update that element (start
count from 1 noy from 0) - ");
scanf(" %d", &i);
i = i - 1;
if(i <= index && i > -1)
{
printf("\nEnter value to update this imdex's element - ");
scanf(" %d", &array[i]);
}
else
printf("\nindex is outof range !\n");
break;
case 3: printf("\nArray =>> ");
if(index == -1)
printf("Nothig to show !\n");
else
{
for(i = 0; i<= index; i++)
printf("%d ", array[i]);
printf("\n");
}
break;
case 4: return;
```

```
default : printf("\Enter proper number from given option !\n");
}
}
}
```

OUTPUT =>

```
// Problem Statement – to delete element form an array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
void main()
{
int array[50], index = 49;
int i,j;
for(i = 0; i<50; i++)
{
array[i] = i+1;
}
printf("\nThere is a array of 50 numbers .....\n");
printf("\nArray =>> ");
for(i = 0; i< 50; i++)
printf("%d ", array[i]);
printf("\n");
while(1)
{
printf("\n1. delete a element form array\n2. update a element\n3. display
array\n4. exit()\nEnter Your choice - ");
scanf(" %d", &i);
switch(i)
{
case 1: if(index < 0)
{
printf("\nArray is empty !\n");
break;
}
printf("\nEnter value of index to delete that element (start count from 1 noy from 0)-");
scanf(" %d", &i);
i = i - 1;
if(i <= index && i > -1)
{
for(j = 0; j != i; j++);
while(j != index)
{
j++;
array[j-1] = array[j];
}
index--;
}
else
printf("\nindex is outof range !\n");
break;
case 2: printf("\nEnter value of index to update that element(start count from 1 noy from
0)-");
scanf(" %d", &i);
i = i - 1;
if(i <= index && i > -1)
{
printf("\nEnter value to update this imdex's element - ");
```

```
scanf(" %d", &array[i]);
} else
printf("\nindex is outof range !\n");
break;
case 3: printf("\nArray =>> ");
if(index == -1)
printf("Nothig to show !\n");
else
{
for(i = 0; i<= index; i++)
printf("%d ", array[i]);
printf("\n");
}
break;
case 4: return;
default : printf("\Enter proper number from given option !\n");
}
}
}
```

OUTPUT=>

```
// Problem Statement - merge two sorted list using an array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
int *array1 = NULL, *array2 = NULL, *mergeArray = NULL;
void display(int *array)
{
int i;
for(i =1; i<array[0]; i++)
printf("%d ", array[i]);
printf("\n");
}
void gets()
{
char x;
scanf("%c", &x);
while(x != '\n')
scanf("%c", &x);
}
void main()
{
int length;
int i;
printf("\nEnter length of first Array - ");
scanf(" %d", &length);
if(length<1)
{
printf("\nValue less than 1 is not valid ! Program terminated\n");
return;
}
array1 = (int *)malloc((length + 1) * sizeof(int));
if(array1 == NULL)
{
printf("\nError occured during memory allcation ...\nProgram terminated !\n");
return;
}
array1[0] = length+1;
```

```
printf("\nFill this first array with values in sorted order \n");
for(length = 1; length< array1[0]; length++)
scanf(" %d", &array1[length]);
gets();
printf("\nEnter length of second Array - ");
scanf(" %d", &length);
if(length<1)
{
printf("\nValue less than 1 is not valid ! Program terminated\n");
return;
}
array2 = (int *)malloc((length + 1) * sizeof(int));
if(array2 == NULL)
{
printf("\nError occured during memory allcation ...\nProgram terminated !\n");
return;
}
array2[0] = length+1;
printf("\nFill this second array with values in sorted order(sorting type must be same as
first array (both
must be assending or decending))\n");
for(length = 1; length< array2[0]; length++)
scanf(" %d", &array2[length]);
gets();
mergeArray = (int *)malloc((array1[0] + array2[0] - 1) * sizeof(int));
if(mergeArray == NULL)
{
printf("\nError occured during memory allcation ...\nProgram terminated !\n");
return;
}
mergeArray[0] = (array1[0] + array2[0] - 1);
length = 1;
i = 1;
for(; length < array1[0]; length++)
{
while(array1[length]>array2[i] && i < array2[0])
{
mergeArray[length + i -1] = array2[i];
i++;
}
mergeArray[length + i -1] = array1[length];
}
while( i < array2[0])
{
mergeArray[length + i -1] = array2[i];
i++;
}
printf("\nFirst Array => ");
display(array1);
printf("\nSecond Array => ");
display(array2);
printf("\nMerge Array => ");
display(mergeArray);
}
```


OUTPUT=>

```
// Problem Statement – search the given element using Linear search technique
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
```

```
#include<stdio.h>
#include<malloc.h>
void gets()
{
char x;
scanf("%c", &x);
while(x != '\n')
scanf("%c", &x);
}
void main()
{
int *array, index, length;
int i;
char ans;
printf("\nHow many terms would be in the array - ");
scanf(" %d", &length);
gets();
if(length < 1)
{
printf("\nValue less than 1 is not valid ! Program Terminated !\n");
return;
}
array = (int *)malloc(length * sizeof(int));
if(array == NULL)
{
printf("\nError occured during memory allocation !. Program terminated !\n");
return;
}
printf("\nFill the array with value .....\n");
for(index = 0; index<length; index++)
scanf("%d", &array[index]);
gets();
printf("\nComplete !\n\n");
printf("\nYour Array => ");
for(index=0; index<length; index++)
printf("%d ", array[index]);
loop:
printf("\n\nEnter a value to search if this value is present in array or not \nEnter your
value - ");
scanf(" %d", &i);
for(index = 0; index < length; index++)
{
if(array[index] == i)
break;
}
if(index == length)
printf("\nThe value you are searching for is not present in this list !\n");
else
printf("\nValue found in this list ! Index no is %d\n", index);
printf("\nDo You want to search another number ( Y or N ) ? -> ");
gets();
scanf("%c", &ans);
if(ans == 'Y' || ans == 'y')
goto loop;
printf("\nEnd of Program !\n");
}
```

OUTPUT=>

```
// Problem Statement – search the given element using Binary search technique
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
int *array, arrayIndex, length;
void gets()
{
char x;
scanf("%c", &x);
while(x != '\n')
scanf("%c", &x);
} int binarySearch(int value, int start, int end)
{
arrayIndex = (start + end)/2;
if(array[arrayIndex] == value)
{
printf("\nValue found ! index=> %d\n", arrayIndex);
return(1);
}
if( start == end || start < 0 || end >= length)
return 0;
if(array[arrayIndex] > value)
{
if(array[0] < array[length-1])
end = arrayIndex-1;
else
start = arrayIndex+1;
}
else
{
if(array[0] < array[length-1])
start = arrayIndex+1;
else
end = arrayIndex-1;
}
return(binarySearch(value, start, end));
}
void main()
{
int i;
char ans;
printf("\nHow many terms would be in the array - ");
scanf(" %d", &length);
gets();
if(length < 1)
{
printf("\nValue less than 1 is not valid ! Program Terminated !\n");
return;
}
array = (int *)malloc(length * sizeof(int));
if(array == NULL)
{
printf("\nError occured during memory allocation !. Program terminated !\n");
return;
}
printf("\nFill the array with value in sorted order .....\n");
for(arrayIndex = 0; arrayIndex<length; arrayIndex++)
```

```
scanf("%d", &array[arrayIndex]);
gets();
printf("\nComplete !\n\n");
printf("\nYour Array => ");
for(arrayIndex=0; arrayIndex<length; arrayIndex++)
printf("%d ", array[arrayIndex]);
loop:
printf("\n\nEnter a value to search if this value is present in array or not \nEnter your
value - ");
scanf(" %d", &i);
if(!binarySearch(i, 0, length-1))
printf("\nThe value you are searching for is not present in this list !\n");
printf("\nDo You want to search another number ( Y or N ) ? -> ");
gets();
scanf("%c", &ans);
if(ans == 'Y' || ans == 'y')
goto loop;
printf("\nEnd of Program !\n");
}
```

OUTPUT=>

```
// Problem Statement – search the given element using Interpolation search technique
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
int *array, arrayIndex, length;
void gets()
{
char x;
scanf("%c", &x);
while(x != '\n')
scanf("%c", &x);
} int interpolationSearch(int value, int start, int end)
{
arrayIndex = start + ((end - start) /(float) (array[end] - array[start]) * (value - array[start]));
if(arrayIndex >= length)
return 0;
if(array[arrayIndex] == value)
{
printf("\nValue found ! index=> %d\n", arrayIndex);
return(1);
}
if(array[arrayIndex] > value)
{
if(array[0] < array[length-1])
end = arrayIndex-1;
else
start = arrayIndex+1;
return(interpolationSearch(value, start, end));
}
else
{
if(array[0] < array[length-1])
start = arrayIndex+1;
else
end = arrayIndex-1;
return(interpolationSearch(value, start, end));
}
```

```
return 0;
}
void main()
{
int i;
char ans;
printf("\nHow many terms would be in the array - ");
scanf(" %d", &length);
gets();
if(length < 1)
{
printf("\nValue less than 1 is not valid ! Program Terminated !\n");
return;
}
array = (int *)malloc(length * sizeof(int));
if(array == NULL)
{
printf("\nError occured during memory allocation !. Program terminated !\n");
return;
}
printf("\nFill the array with value in sorted and distributed manner .....\n");
for(arrayIndex = 0; arrayIndex<length; arrayIndex++)
scanf("%d", &array[arrayIndex]);
gets();
printf("\nComplete !\n\n");
printf("\nYour Array => ");
for(arrayIndex=0; arrayIndex<length; arrayIndex++)
printf("%d ", array[arrayIndex]);
loop:
printf("\n\nEnter a value to search if this value is present in array or not \nEnter your
value - ");
scanf(" %d", &i);
if(!interpolationSearch(i, 0, length-1))
printf("\nThe value you are searching for is not present in this list !\n");
printf("\nDo You want to search another number ( Y or N ) ? -> ");
gets();
scanf("%c", &ans);
gets();
if(ans == 'Y' || ans == 'y')
goto loop;
printf("\nEnd of Program !\n");
}
```

OUTPUT=>

```
//problem statement – Addition of two Polynomial using an Array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
void main()
{
int p1[20][2], p2[20][2], sum[40][2];
int l1,l2,l3;
int i,j;
printf("Both of two polynomial will be function of x ( f(x) )\n");
printf("\nEnter number of terms in first polynomial(number of terms mustbe less than or
equal to 20)- ");
scanf(" %d", &l1);
printf("\nEnter data in first polynomial -\n");
for(i=0; i<l1; i++)
```

```
{
printf("for term no %d\n", i+1);
printf("coefficient -> ");
scanf(" %d", &p1[i][0]);
printf("power -> ");
scanf(" %d", &p1[i][1]);
}
printf("\n");
printf("\nEnter number of terms in second polynomial(number mustbe less than or equal
to 20) - ");
scanf(" %d", &l2);
printf("\nEnter data in second polynomial -\n");
for(i=0; i<l2; i++)
{
printf("for term no %d\n", i+1);
printf("coefficient -> ");
scanf(" %d", &p2[i][0]);
printf("power -> ");
scanf(" %d", &p2[i][1]);
}
printf("\n");
l3=-1;
for(i=0;i<l1;i++)
{
l3++;
sum[l3][0] = p1[i][0];
sum[l3][1] = p1[i][1];
}
for(i=0;i<l2;i++)
{
for(j=0;j<=l3;j++)
{
if(p2[i][1] == sum[j][1])
{
sum[j][0] = sum[j][0] + p2[i][0];
break;
}
} if(j>l3)
{
l3++;
sum[l3][0] = p2[i][0];
sum[l3][1] = p2[i][1];
}
}
printf("\nResults =>>\n");
printf("Polynomial 1 =>> ");
for(i=0;i<l1;i++)
{
if(p1[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", p1[i][0], 'x', p1[i][1]);
}
printf("\n");
printf("Polynomial 2 =>> ");
for(i=0;i<l2;i++)
{
if(p2[i][0] > -1)
```

```
printf("+");
printf("%d%c^(%d) ", p2[i][0], 'x', p2[i][1]);
}
printf("\n");
printf("Sum of Polynomial 1 and 2 =>> ");
for(i=0;i<=l3;i++)
{
if(sum[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", sum[i][0], 'x', sum[i][1]);
}
printf("\n");
}
```

OUTPUT=>

```
//problem statement - multiplication of two polynomial using an array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
void main()
{
int p1[20][2], p2[20][2], mul[400][2];
int l1,l2,l3;
int i,j,k;
int v;
printf("Both of two polynomial will be function of x ( f(x) )\n");
printf("\nEnter number of terms in first polynomial (number mustbe less than or equal to
20) - ");
scanf(" %d", &l1);
printf("\nEnter data in first polynomial -\n");
for(i=0; i<l1; i++)
{
printf("for term no %d\n", i+1);
printf("coefficient -> ");
scanf(" %d", &p1[i][0]);
printf("power -> ");
scanf(" %d", &p1[i][1]);
}
printf("\n");
printf("\nEnter number of terms in second polynomial (number mustbe less than or
equal to 20) - ");
scanf(" %d", &l2);
printf("\nEnter data in second polynomial -\n");
for(i=0; i<l2; i++)
{
printf("for term no %d\n", i+1);
printf("coefficient -> ");
scanf(" %d", &p2[i][0]);
printf("power -> ");
scanf(" %d", &p2[i][1]);
}
printf("\n");
l3=0;
for(i=0;i<l1;i++)
{
for(j=0;j<l2;j++)
{
v = p1[i][1] + p2[j][1];
for(k=0;k<l3;k++)
```

```
{
if(v == mul[k][1])
{
mul[k][0] = mul[k][0] + p1[i][0] * p2[j][0];
break;
}
} if(!(k<l3))
{
mul[k][0] = p1[i][0] * p2[j][0];
mul[k][1] = p1[i][1] + p2[j][1];
l3++;
}
}
}
printf("\nResults =>>\n");
printf("Polynomial 1 =>> ");
for(i=0;i<l1;i++)
{
if(p1[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", p1[i][0], 'x', p1[i][1]);
}
printf("\n");
printf("Polynomial 2 =>> ");
for(i=0;i<l2;i++)
{
if(p2[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", p2[i][0], 'x', p2[i][1]);
}
printf("\n");
printf("Multiplication of Polynomial 1 and 2 =>> ");
for(i=0;i<l3;i++)
{
if(mul[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", mul[i][0], 'x', mul[i][1]);
}
printf("\n");
}
```

OUTPUT=>

```
//problem statement – transform Sparse matrix into Compact matrix using array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
void main()
{
int **p, **q;
int r,c;
int i,k,sum=0;
printf("\nEnter row and column number of matrix - ");
scanf(" %d %d", &r, &c);
p = (int **)malloc(c*sizeof(int*));
if(p == NULL)
{
printf("\nError occured during memory allocation ! Program terminated !\n");
return;
```

```
}
for(i =0; i<c;i++)
{
p[i] = (int *)malloc(r*sizeof(int));
if(p[i] == NULL)
{
printf("\nError occured during memory allocation ! Program terminated !\n");
return;
}
}
printf("\nFill the matrix with value \n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
scanf(" %d", &p[i][j]);
}
printf("\n");
k = 0;
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
if(p[i][j] != 0)
k++;
}
q = (int **)malloc(k*sizeof(int*));
if(q == NULL)
{
printf("\nError occured during memory allocation ! Program terminated !\n");
return;
}
for(i =0; i< k;i++)
{
q[i] = (int *)malloc(3*sizeof(int));
if(q[i] == NULL)
{
printf("\nError occured during memory allocation ! Program terminated !\n");
return;
}
}
count =0;
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
if(p[i][j] != 0)
{
q[count][0] = i;
q[count][1] = j;
q[count][2] = p[i][j];
count++;
}
}
}
printf("\nSparse Matrix =>\n");
for(i = 0; i < r; i++)
{
for(j =0; j<c; j++)
```

```
printf(" %d", p[i][j]);
printf("\n");
}
printf("\n");
printf("\nCompact Matrix =>\n");
for(i = 0; i < 3; i++)
{
for(j =0; j<k; j++)
printf(" %d", q[j][i]);
printf("\n");
}
printf("\n");
}
```

OUTPUT=>

```
// Problem Statement - Single Link List
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
struct node
{
int value;
struct node *next;
};
typedef struct node node;
//node type pointer to keep track
node *start = NULL, *end = NULL, *temp = NULL;
//function to operate single link list
void add_last();
void add_first();
```

```
void add_pos();
void del_last();
void del_first();
void del_pos();
void update();
void display();
//definations
void add_last()
{
temp = (node *)malloc(sizeof(node));
if(temp == NULL)
{
printf("\nError occured during Memory allocation !\nNew node creation
failed !\n");
return;
}
printf("\nEnter value to initialize this new node -> ");
scanf("%d", &temp->value);
if(start == NULL)
{
start = end = temp;
temp->tail = NULL;
} else
{
end->tail = temp;
end = temp;
temp->tail = NULL;
} temp =
NULL;
printf("\n*** New node has added at last position in Double link list ! ");
}
void add_first()
{
temp = (node *)malloc(sizeof(node));
if(temp == NULL)
{
printf("\nError occured during Memory allocation !\nNew node creation failed ! \
n");
return;
}
printf("\nEnter value to initialize this new node -> ");
scanf("%d", &temp->value);
if(start == NULL)
{
start = end = temp;
temp->tail = NULL;
} else
{
temp->tail = start;
start = temp;
} temp =
NULL;
printf("\n*** New node has added at first position in Double link list ! ");
}
void add_pos()
{
int c, i;
```

```
node *temp2, *temp3;
temp2 = (node *)malloc(sizeof(node));
if(temp2 == NULL)
{
printf("\nError occured during Memory allocation !\nNew node
creation failed ! \n");
return;
}
printf("\nEnter value to initialize this new node -> ");
scanf("%d", &temp2->value);
//count will start from 1 not 0
printf("\nin which position you want to insert this new node (count start from 1 not from
0)
--> ");
scanf("%d", &c);
i=0;
temp3 = NULL;
for(temp = start; temp != NULL; temp = temp->tail)
{
i++;
if(i==c)
break;
temp3 = temp;
}
if(i != c && temp == NULL)
{
printf("\ngiven position does not exist in this double link list ... \n");
return;
}
if(temp3 == NULL && i == c)
{
temp2->tail = start;
start = temp2;
} else
{
temp2->tail = temp;
temp3->tail = temp2;
}
printf("\n*** New node has been addede successfully !\n");
temp = NULL;
}
void del_last()
{
if(start == NULL)
{
printf("\nThere is nothing to delete !\n");
return;
}
if(start->tail == NULL)
{
free(start);
start = end = NULL;
} else
{
for(temp = start; temp->tail != end; temp = temp->tail);
temp->tail = NULL;
free(end);
```

```
end = temp;
}
printf("\n*** the node in last position has been deleted successfully !\n");
temp = NULL;
}
void del_first()
{
if(start == NULL)
{
printf("\nThere is nothing to delete !\n");
return;
} if(start->tail == NULL)
{
free(start);
start = end = NULL;
} else
{
temp = start;
start = start->tail;
free(temp);
}
printf("\n*** the node in first position has been deleted successfully !\n");
temp = NULL;
}
void del_pos()
{
int c, i;
node *temp2;
if(start == NULL)
{
printf("\nThere is nothing to delete !\n");
return;
}
printf("\nEnter the position from where you want to delete a node - ");
scanf("%d", &c);
if(start->tail == NULL)
{
free(start);
start = end = NULL;
printf("\n*** The node in %d position has been deleted
successfully !\n", c);
return;
} else if(c == 1)
{
temp = start;
start = start->tail;
free(temp);
printf("\n*** The node in %d position has been deleted successfully !\n", c);
return;
}i
= 0;
c = c-1;
for(temp = start; temp != NULL; temp = temp->tail)
{
i++;
if(i == c)
{
```

```
break;
}
} if(i
!= c
&& temp == NULL)
{
printf("\nGiven position does not exist in this link list .... \n");
return;
} temp2 =
temp->tail;
temp->tail = temp2->tail;
free(temp2);
printf("\n*** The node in %d position has been deleted successfully !\
n", c);
temp = NULL;
}
void update()
{
int c, i;
if(start == NULL)
{
printf("\nThere is nothing to update ...\n");
return;
}
printf("\nEnter node position to update this (count start from 1 not from 0)-> ");
scanf("%d", &c);
i = 0;
for(temp = start; temp != NULL; temp = temp->tail)
{
i++;
if(c == i)
{
break;
}
}
if(c != i && temp == NULL)
{
printf("\nThis position is not exist in this link list ... \n");
return;
}i
= temp->value;
printf("\nNode found. Enter a value to update this node - ");
scanf("%d", &temp->value);
printf("\n*** Completed !\n%d position's node has been updated from %d to %d
successfully !\n", c, i, temp->value);
temp = NULL;
}
void display()
{
printf("\n                    \nDOUBLE LINK LIST\
n_____\n=>> ");
if(start == NULL)
{
printf("Link List is Empty !\n");
}
for(temp = start; temp != NULL; temp = temp->tail)
{
```

```
printf("%d ", temp->value);
}
printf("\n\n");
}
void main()
{
int choice;
while(1)
{
printf("\n\n1. Add a node in last position\n2. Add a node in first position\n3. Add a
node in between the link list\n\n4. Delete a node from last position\n5. Delete a
node from first position\n6. Delete a node from in between the link list\n\n7. update
a node's value\n8. Display all nodes and values\n9. Exit( )\n\nEnter your choice-");
fflush(stdin);
scanf("%d",&choice);
switch(choice)
{
case 1:add_last();
break;
case 2:add_first();
break;
case 3:add_pos();
break;
case 4:del_last();
break;
case 5:del_first();
break;
case 6:del_pos();
break;
case 7:update();
break;
case 8:display();
break;
case 9:printf("\n\nExit status * Successfully !\n");
return;
default: printf("\n\nInvalid Input !!!!!! Please choose a proper option
and enter that number correctly ....\n");
break;
}
}
}
// Problem Statement - Single Link List
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
struct node
{
int coefficient;
int power;
struct node *next;
};
typedef struct node node;
struct polynomial
{
char functionOf;
node *start;
```

```
};
typedef struct polynomial polynomial;
int addNode(node **start, char ch); // return 1 means success 0 means fail
polynomial *equations = NULL;
int eqsNo; // how many equations
void makeEquation();
void displayEquations();
void addition();
void subtraction();
int addNode(node **start, char ch)
{
node *newNode, *temp;
newNode = (node *)malloc(sizeof(node));
if(newNode == NULL)
{
printf("\nError Occured during MEMORY allocation for a new
node..... ! ");
exit(0);
}
printf("\nEnter value of coefficient of %c - ", ch);
scanf(" %d", &newNode->coefficient);
printf("\nEnter value of power of %c - ", ch);
scanf(" %d", &newNode->power);
if(*start == NULL)
{
newNode->next = NULL;
*start = newNode;
return 1;
}
for(temp = *start; temp->next != NULL; temp = temp->next);
newNode->next = NULL;
temp->next = newNode;
return 1;
}
void makeEquation()
{
int i, j, k;
printf("\nHow many equation do you need - ");
scanf(" %d", &eqsNo);
if(eqsNo < 1)
{
printf("\nNumbers smaller than 1 is not acceptable in this case !
program terminated !\n");
exit(0);
} equations =
(
polynomial *)malloc(eqsNo *
sizeof(polynomial));
if(equations == NULL)
{
printf("\Error occured during MEMORY allocation for array ..... ");
exit(0);
}
for(i = 0; i < eqsNo; i++)
{
equations[i].start = NULL;
printf("\nNo. %d equation is polynomial of ( x or y or something else, Enter
```

```
that character) - " , i+1);
scanf(" %c", &equations[i].functionOf);
printf("\nHow many terms are there in equation no %d - ",i+1);
scanf(" %d", &j);
if(j < 1)
{
printf("\nNumbers smaller than 1 is not acceptable in this
case ! program terminated !\n");
exit(0);
}
k = 0;
for( ; j > 0; j--)
{
k++;
printf("\nfor term %d ---", k);
if(addNode( &equations[i].start, equations[i].functionOf ) == 0)
{
printf("\nTry again .....");
j++;
continue;
}
}
}
}
void displayEquations()
{
int i;
for(i = 0; i<eqsNo; i++)
{
node *temp;
temp = equations[i].start;
if(temp == NULL)
{
printf("\nEquatio no %d is empty .....", i+1);
continue;
}
printf("\nEquation %d ==>>\n", i+1);
while(temp != NULL)
{
printf("%d%c^(%d) ", temp->coefficient, equations[i].functionOf,
temp->power);
temp = temp->next;
if(temp != NULL)
{
if(temp->coefficient > -1)
{
printf("+");
}
}
}
printf("\n");
}
}
void addition()
{
int pno[2];
int i,c[2];
```

```
node *temp, *newNode, *temp2;
polynomial p1;
p1.start = NULL;
printf("\nEnter any two equation No (like 1, 2, or 3, or somthing else if there...) - ");
scanf(" %d %d", &pno[0], &pno[1]);
fflush(stdin);
if(pno[0] > eqsNo || pno[1] > eqsNo || pno[0] < 1 || pno[1] < 1)
{
printf("\nYou Enter invalid equation no .....! Addition terminated !\n");
exit(0);
}
for(i = 0; i < 2; i++)
{
temp = equations[pno[i]-1].start;
c[i] = 0;
while(temp != NULL)
{
c[i] = c[i]++;
temp = temp->next;
}
} if(equations[pno[0]-1].functionOf != equations[pno[1]-1].functionOf)
{
printf("\nEquation No %d and %d are polynomial of two different functions !\
nSum of this two polynomial would be function of ' f(%c%c) ' \nResult => "pno[0],
pno[1], equations[pno[0]-1].functionOf, equations[pno[1]-1].functionOf);
temp = equations[pno[0]-1].start;
while(temp != NULL)
{
printf("%d%c^(%d) ", temp->coefficient, equations[pno[0]-
1].functionOf, temp->power);
temp = temp->next;
if(temp != NULL && temp->coefficient >= 0)
{
printf("+");
}
}
temp = equations[pno[1]-1].start;
while(temp != NULL)
{
if(temp->coefficient >0)
{
printf("+");
}
printf("%d%c^(%d) ", temp->coefficient, equations[pno[1]-
1].functionOf, temp->power);
temp = temp->next;
}
printf("\n\n");
return;
} else
{
temp = equations[pno[0]-1].start;
while(temp != NULL)
{
//node *newNode, *temp2;
newNode = (node *)malloc(sizeof(node));
if(newNode == NULL)
```

```
{
printf("\nError Occured during MEMORY allocation
for a new node..... ! ");
exit(0);
}
newNode->coefficient = temp->coefficient;
newNode->power = temp->power;
newNode->next = NULL;
if(p1.start == NULL)
{
p1.start = newNode;
} else
{
for(temp2 = p1.start; temp2->next != NULL; temp2 =
temp2->next);
temp2->next = newNode;
}
temp = temp->next;
}
temp = equations[pno[1]-1].start;
while(temp != NULL)
{
temp2 = p1.start;
while(temp2 != NULL)
{
if(temp->power == temp2->power)
{
//add in-side node
temp2->coefficient = temp2->coefficient +
temp->coefficient;
//temp2->power = temp2->power + temp->power;
break;
}
temp2 = temp2->next;
}
if(temp2 == NULL)
{
//add a extra node
newNode = (node *)malloc(sizeof(node));
if(newNode == NULL)
{
printf("\nError Occured during MEMORY
allocation for a new node..... ! ");
exit(0);
}
newNode->coefficient = temp->coefficient;
newNode->power = temp->power;
newNode->next = NULL;
for(temp2 = p1.start; temp2->next != NULL;
temp2 = temp2->next);
temp2->next = newNode;
temp2 = NULL;
}
temp = temp->next;
}
}
printf("\nEquation No %d and %d are polynomial of two same functions !\nSum of
```

```
this two polynomial would be function of ' f(%c) ' \nResult => ", pno[0],
pno[1], equations[pno[0]-1].functionOf );
temp2 = p1.start;
while(temp2 != NULL)
{
printf("%d%c^(%d) ", temp2->coefficient, equations[pno[1]- 1].functionOf,
temp2->power);
temp2 = temp2->next;
if(temp2 != NULL && temp2->coefficient >= 0)
{
printf("+");
}
}
printf("\n\n");
}
void subtraction()
{
int pno[2];
int i,c[2];
node *temp, *newNode, *temp2;
polynomial p1;
p1.start = NULL;
printf("\nEnter any two equation No (like 1, 2, or 3, or somthing else if there...) - ");
scanf(" %d %d", &pno[0], &pno[1]);
fflush(stdin);
if(pno[0] > eqsNo || pno[1] > eqsNo || pno[0] < 1 || pno[1] < 1)
{
printf("\nYou Enter invalid equation no .....! Addition terminated !\n");
exit(0);
}
for(i = 0; i < 2; i++)
{
temp = equations[pno[i]-1].start;
c[i] = 0;
while(temp != NULL)
{
c[i] = c[i]++;
temp = temp->next;
}
} if(equations[pno[0]-1].functionOf != equations[pno[1]-1].functionOf)
{
printf("\nEquation No %d and %d are polynomial of two different functions !\
nSum of this two polynomial would be function of ' f(%c%c) ' \nResult => ",
pno[0], pno[1], equations[pno[0]-1].functionOf, equations[pno[1]-1].functionOf);
temp = equations[pno[0]-1].start;
while(temp != NULL)
{
printf("%d%c^(%d) ", temp->coefficient, equations[pno[0]-
1].functionOf, temp->power);
temp = temp->next;
if(temp != NULL && temp->coefficient >= 0)
{
printf("+");
}
}
temp = equations[pno[1]-1].start;
while(temp != NULL)
```

```
{
if(-(temp->coefficient) >= 0)
{
printf("+");
}
printf("%d%c^(%d) ", -(temp->coefficient),
equations[pno[1]-1].functionOf, temp->power);
temp = temp->next;
}
printf("\n\n");
return;
} else
{
temp = equations[pno[0]-1].start;
while(temp != NULL)
{
//node *newNode, *temp2;
newNode = (node *)malloc(sizeof(node));
if(newNode == NULL)
{
printf("\nError Occured during MEMORY allocation
for a new node..... ! ");
exit(0);
}
newNode->coefficient = temp->coefficient;
newNode->power = temp->power;
newNode->next = NULL;
if(p1.start == NULL)
{
p1.start = newNode;
} else
{
for(temp2 = p1.start; temp2->next != NULL; temp2 =
temp2->next);
temp2->next = newNode;
}
temp = temp->next;
}
temp = equations[pno[1]-1].start;
while(temp != NULL)
{
temp2 = p1.start;
while(temp2 != NULL)
{
if(temp->power == temp2->power)
{
//add in-side node
temp2->coefficient = temp2->coefficient– temp->coefficient;
break;
}
temp2 = temp2->next;
}
if(temp2 == NULL)
{
//add a extra node
newNode = (node *)malloc(sizeof(node));
if(newNode == NULL)
```

```
{
printf("\nError Occured during MEMORY
allocation for a new node..... ! ");
exit(0);
}
newNode->coefficient = -temp->coefficient;
newNode->power = temp->power;
newNode->next = NULL;
for(temp2 = p1.start; temp2->next != NULL;
temp2 = temp2->next);
temp2->next = newNode;
temp2 = NULL;
}
temp = temp->next;
}
}
printf("\nEquation No %d and %d are polynomial of two same functions !\nSum of this
two polynomial would be function of ' f(%c) ' \nResult => ", pno[0], pno[1],
equations[pno[0]-1].functionOf );
temp2 = p1.start;
while(temp2 != NULL)
{
printf("%d%c^(%d) ", temp2->coefficient, equations[pno[1]- 1].functionOf,
temp2->power);
temp2 = temp2->next;
if(temp2 != NULL && temp2->coefficient >= 0)
{
printf("+");
}
}
printf("\n\n");
}
void main()
{
int i;
while(1)
{
printf("\nEnter Your Choice from below Options -\n1. create and initialize
Equation\n2. display equations\n3. add any two equation\ n4. subtract one
equation from another\n5. delete all equations and recreate equations\n6. Exit()\n\
nEnter your choice -> ");
scanf(" %d", &i);
switch(i)
{
case 1: makeEquation();
break;
case 2: displayEquations();
break;
case 3: addition();
break;
case 4: subtraction();
break;
case 5: free(equations);
makeEquation();
break;
case 6: return;
break;
```

```
default : printf("\nYou enter a wrong option , please enter a
right option .....\n");
break;
}
}
}
// Problem Statement - Circular(Single) Link List
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
struct node
{
int value;
struct node *tail;
};
typedef struct node node;
//node type pointer to keep track
node *start = NULL, *end = NULL, *temp = NULL;
//function to operate single link list
void add_last();
void add_first();
void add_pos();
void del_last();
void del_first();
void del_pos();
void update();
void display();
void add_last()
{
temp = (node *)malloc(sizeof(node));
if(temp == NULL)
{
printf("\nError occured during Memory allocation !\nNew node
creation failed ! \n");
return;
}
printf("\nEnter value to initialize this new node -> ");
scanf("%d", &temp->value);
if(start == NULL)
{
start = end = temp;
temp->tail = start;
} else
{
end->tail = temp;
end = temp;
temp->tail = start;
}
temp = NULL;
printf("\n*** New node has added at last position in Double link list ! ");
}
void add_first()
{
temp = (node *)malloc(sizeof(node));
if(temp == NULL)
{
printf("\nError occured during Memory allocation !\nNew node
```

```
creation failed ! \n");
return;
}
printf("\nEnter value to initialize this new node -> ");
scanf("%d", &temp->value);
if(start == NULL)
{
start = end = temp;
temp->tail = start;
} else
{
temp->tail = start;
start = temp;
end->tail = start;
} temp =
NULL;
printf("\n*** New node has added at first position in Double link list ! ");
}
void add_pos()
{
int c, i;
node *temp2, *temp3;
temp2 = (node *)malloc(sizeof(node));
if(temp2 == NULL)
{
printf("\nError occured during Memory allocation !\nNew node
creation failed ! \n");
return;
}
printf("\nEnter value to initialize this new node -> ");
scanf("%d", &temp2->value);
//count will start from 1 not 0
printf("\nin which position you want to insert this new node (count start from 1 not from
0)
--> ");
scanf("%d", &c);
i=0;
temp3 = NULL;
for(temp = start; ; temp = temp->tail)
{
i++;
if(i==c)
break;
temp3 = temp;
if(temp == end )
break;
}
if(i != c && temp == start)
{
printf("\ngiven position does not exist in this double link list ... \n");
return;
}
if(temp3 == NULL && i == c)
{
temp2->tail = start;
start = temp2;
end->tail = start;
```

```
} else
{
temp2->tail = temp;
temp3->tail = temp2;
}
printf("\n*** New node has been addede successfully !\n");
temp = NULL;
}
void del_last()
{
if(start == NULL)
{
printf("\nThere is nothing to delete !\n");
return;
}
if(start->tail == NULL)
{
free(start);
start = end = NULL;
} else
{
for(temp = start; temp->tail != end; temp = temp->tail);
temp->tail = NULL;
free(end);
end = temp;
}
printf("\n*** the node in last position has been deleted successfully !\n");
temp = NULL;
}
void del_first()
{
if(start == NULL)
{
printf("\nThere is nothing to delete !\n");
return;
} if(start->tail == NULL)
{
free(start);
start = end = NULL;
} else
{
temp = start;
start = start->tail;
end->tail = start;
free(temp);
}
printf("\n*** the node in first position has been deleted successfully !\n");
temp = NULL;
}
void del_pos()
{
int c, i;
node *temp2;
if(start == NULL)
{
printf("\nThere is nothing to delete !\n");
return;
```

```
}
printf("\nEnter the position from where you want to delete a node - ");
scanf("%d", &c);
if(start->tail == NULL)
{
free(start);
start = end = NULL;
printf("\n*** The node in %d position has been deleted successfully !\n", c);
return;
} else if(c
== 1)
{
temp = start;
start = start->tail;
end->tail = start;
free(temp);
printf("\n*** The node in %d position has been deleted successfully !\n", c);
return;
}i
= 0;
c = c-1;
for(temp = start; ; temp = temp->tail)
{
i++;
if(i == c)
{
break;
}
if(temp == end )
break;
} if(i
!= c
&& temp == start)
{
printf("\nGiven position does not exist in this link list .... \n");
return;
} temp2 =
temp->tail;
temp->tail = temp2->tail;
free(temp2);
printf("\n*** The node in %d position has been deleted successfully !\
n", c);
temp = NULL;
}
void update()
{
int c, i;
if(start == NULL)
{
printf("\nThere is nothing to update ...\n");
return;
}
printf("\nEnter node position to update this (count start from 1 not from
0)-> ");
scanf("%d", &c);
i = 0;
for(temp = start; ; temp = temp->tail)
```

```
{
i++;
if(c == i)
{
break;
}
if(temp == end )
break;
}
if(c != i && temp == NULL)
{
printf("\nThis position is not exist in this link list ... \n");
return;
}i
= temp->value;
printf("\nNode found. Enter a value to update this node - ");
scanf("%d", &temp->value);
printf("\n*** Completed !\n%d position's node has been updated from
%d to %d successfully !\n", c, i, temp->value);
temp = NULL;
}
void display()
{
printf("\n_________________\nDOUBLE LINK LIST\
n_________________\n=>> ");
if(start == NULL)
{
printf("Link List is Empty !\n");
}
for(temp = start; ; temp = temp->tail)
{
printf("%d ", temp->value);
if(temp == end )
break;
}
printf("\n\n");
}
void main()
{
int choice;
while(1)
{
printf("\n\n1. Add a node in last position\n2. Add a node in first position\n3. Add a
node in between the link list\n\n4. Delete a node from last position\n5. Delete a
node from first position\n6. Delete a node from in between the link list\n\n7. update
a node's value\n8. Display all nodes and values\n9. Exit( )\n\nEnter your choice -");
fflush(stdin);
scanf("%d",&choice);
switch(choice)
{
case 1:add_last();
break;
case 2:add_first();
break;
case 3:add_pos();
break;
case 4:del_last();
```

```
break;
case 5:del_first();
break;
case 6:del_pos();
break;
case 7:update();
break;
case 8:display();
break;
case 9:printf("\n\nExit status * Successfully !\n");
return;
default: printf("\n\nInvalid Input !!!!!! Please choose a
proper option and enter that number
correctly ....\n");
break;
}
}
}
// Problem Statement - Double Link List
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
struct node
{
struct node * head;
int value;
struct node * tail;
};
typedef struct node node;
//node type pointer to keep track
node *start = NULL, *end = NULL, *temp = NULL;
//function to operate double link list
void add_last();
void add_first();
void add_pos();
void del_last();
void del_first();
void del_pos();
void update();
void display();
void add_last()
{
temp = (node *)malloc(sizeof(node));
if(temp == NULL)
{
printf("\nError occured during Memory allocation !\nNew node
creation failed ! \n");
return;
}
printf("\nEnter value to initialize this new node -> ");
scanf("%d", &temp->value);
if(start == NULL)
{
start = end = temp;
temp->head = start;
temp->tail = NULL;
} else
```

```
{
end->tail = temp;
temp->head = end;
end = temp;
temp->tail = NULL;
} temp =
NULL;
printf("\n*** New node has added at last position in Double link list ! ");
}
void add_first()
{
temp = (node *)malloc(sizeof(node));
if(temp == NULL)
{
printf("\nError occured during Memory allocation !\nNew node
creation failed ! \n");
return;
}
printf("\nEnter value to initialize this new node -> ");
scanf("%d", &temp->value);
if(start == NULL)
{
start = end = temp;
temp->head = start;
temp->tail = NULL;
} else
{
temp->tail = start;
start->head = temp;
start = temp;
temp->head = start;
} temp =
NULL;
printf("\n*** New node has added at first position in Double link list ! ");
}
void add_pos()
{
int c, i;
node *temp2;
temp2 = (node *)malloc(sizeof(node));
if(temp2 == NULL)
{
printf("\nError occured during Memory allocation !\nNew node
creation failed ! \n");
return;
}
printf("\nEnter value to initialize this new node -> ");
scanf("%d", &temp2->value);
//count will start from 1 not 0
printf("\nin which position you want to insert this new node (count start from 1 not from
0)
--> ");
scanf("%d", &c);
i=0;
for(temp = start; temp != NULL; temp = temp->tail)
{
i++;
```

```
if(i==c)
break;
}
if(i != c && temp == NULL)
{
printf("\ngiven position does not exist in this double link list ... \
n");
return;
}
temp2->head = temp->head;
temp2->tail = temp;
temp->head = temp2;
temp = temp2->head;
temp->tail = temp2;
printf("\n*** New node has been addede successfully !\n");
temp = NULL;
}
void del_last()
{
if(end == NULL)
{
printf("\nThere is nothing to delete !\n");
return;
} temp =
end;
end = end->head;
end->tail = NULL;
free(temp);
printf("\n*** the node in last position has been deleted successfully !\n");
temp = NULL;
}
void del_first()
{
if(start == NULL)
{
printf("\nThere is nothing to delete !\n");
return;
} temp =
start;
start = start->tail;
free(temp);
printf("\n*** the node in first position has been deleted successfully !\n");
temp = NULL;
}
void del_pos()
{
int c, i;
if(start == NULL)
{
printf("\nThere is nothing to delete !\n");
return;
}
printf("\nEnter the position from where you want to delete a node - ");
scanf("%d", &c);
i = 0;
for(temp = start; temp != NULL; temp = temp->tail)
{
```

```
i++;
if(i == c)
{
break;
}
} if(i
!= c
&& temp == NULL)
{
printf("\nGiven position does not exist in this link list .... \n");
return;
}
temp->head->tail = temp->tail;
temp->tail->head = temp->head;
free(temp);
printf("\n*** The node in %d position has been deleted successfully !\
n", c);
temp = NULL;
}
void update()
{
int c, i;
if(start == NULL)
{
printf("\nThere is nothing to update ...\n");
return;
}
printf("\nEnter node position to update this (count start from 1 not from 0)-> ");
scanf("%d", &c);
i = 0;
for(temp = start; temp != NULL; temp = temp->tail)
{
i++;
if(c == i)
{
break;
}
}
if(c != i && temp == NULL)
{
printf("\nThis position is not exist in this link list ... \n");
return;
}i
= temp->value;
printf("\nNode found. Enter a value to update this node - ");
scanf("%d", &temp->value);
printf("\n*** Completed !\n%d position's node has been updated from
%d to %d successfully !\n", c, i, temp->value);
temp = NULL;
}
void display()
{
printf("\n_____\nDOUBLE LINK LIST\
n_____\n=>> ");
if(start == NULL)
{
printf("Link List is Empty !\n");
```

```
}
for(temp = start; temp != NULL; temp = temp->tail)
{
printf("%d ", temp->value);
}
printf("\n\n");
}
void main()
{
int choice;
while(1)
{
printf("\n\n1. Add a node in last position\n2. Add a node in first position\n3. Add a
node in between the link list\n\n4. Delete a node from last position\n5. Delete a
node from first position\n6. Delete a node from in between the link list\n\n7. update
a node's value\n8. Display all nodes and values\n9. Exit( )\n\nEnter your choice -");
fflush(stdin);
scanf("%d",&choice);
switch(choice)
{
case 1:add_last();
break;
case 2:add_first();
break;
case 3:add_pos();
break;
case 4:del_last();
break;
case 5:del_first();
break;
case 6:del_pos();
break;
case 7:update();
break;
case 8:display();
break;
case 9:printf("\n\nExit status * Successfully !\n");
return;
default: printf("\n\nInvalid Input !!!!!! Please choose a
proper option and enter that number
correctly ....\n");
break;
}
}
}
// Problem Statement - Reverse of Single Link List
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
struct node
{
int value;
struct node *next;
};
typedef struct node node;
node *start = NULL, *end = NULL, *temp1 = NULL, *temp2 = NULL;
```

```
void constructLinkList();
void reverseLinkList();
void display();
void constructLinkList()
{
char ch;
do
{
temp1 = (node *)malloc(sizeof(node));
if(temp1 == NULL)
{
printf("\nError occured during memory allocation for a new
node ... Program terminated !");
exit(0);
}
printf("\nEnter a value for this new node - ");
scanf(" %d", &temp1->value);
temp1->next = NULL;
if(start == NULL)
{
start = end = temp1;
} else
{
end->next = temp1;
end = temp1;
}
printf("\nWant to add another node ? ( y or n) -> ");
scanf(" %c", &ch);
}while(ch == 'Y' || ch == 'y');
}
void reverseLinkList()
{
temp1 = start;
temp2 = end;
while(temp2 != start)
{
for(temp1 = start; temp1->next != temp2; temp1 = temp1->next);
temp2->next = temp1;
temp2 = temp1;
} temp2->next =
NULL;
start = end;
end = temp2;
}
void display()
{
printf("\nLINK LIST ->\n.....\n");
for(temp1 = start; temp1 != NULL; temp1 = temp1->next)
printf("%d ",temp1->value);
printf("\n");
}
void main()
{
constructLinkList();
display();
reverseLinkList();
display();
```

```
}
//problem statement – sparse matrix to compact matrix
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<stdlib.h>
struct array
{
int value;
struct array *next;
};
typedef struct array array;
struct pArray
{
array *value;
struct pArray *next;
};
typedef struct pArray pArray;
void createArray(pArray **start, int r, int c);
void fillArray(pArray *start);
int getValue(pArray *start, int r, int c);
void putValue(pArray *start, int r, int c, int value);
void display(pArray *start);
void createArray(pArray **start, int r, int c)
{
pArray *end, *new;
array *temp1, *temp2;
int a,b;
a = r;
b = c;
new = (pArray *)malloc(sizeof(pArray));
if(new == NULL)
{
printf("\nError Occured during memory allocation ! Program
terminated !\n");
exit(0);
}
new->value = NULL;
new->next = NULL;
*start = new;
end = new;
a--;
while(a)
{
new = (pArray *)malloc(sizeof(pArray));
if(new == NULL)
{
printf("\nError Occured during memory allocation !
Program terminated !\n");
exit(0);
}
new->value = NULL;
new->next = NULL;
end->next = new;
end = new;
a--;
}
end = *start;
```

```
while(end != NULL)
{
end->value = (array *)malloc(sizeof(array));
if(end->value == NULL)
{
printf("\nError Occured during memory allocation !
Program terminated !\n");
exit(0);
}
end->value->next = NULL;
temp2 = end->value;
b--;
while(b)
{
temp1 = (array *)malloc(sizeof(array));
if(temp1 == NULL)
{
printf("\nError Occured during memory allocation !
Program terminated !\n");
exit(0);
} temp1->next =
NULL;
temp2->next = temp1;
temp2 = temp1;
b--;
}
end = end->next;
b = c;
}
}
void fillArray(pArray *start)
{
pArray *end;
array *temp1;
int c;
printf("\nFill the array with values =>\n");
for(end = start; end != NULL; end = end->next)
{
for(temp1 = end->value; temp1 != NULL; temp1 = temp1->next)
{
scanf(" %d", &c);
temp1->value = c;
}
}
}
void putValue(pArray *start, int r, int c, int value)
{
pArray *end;
array *temp1;
int i, j;
for(end = start, i = 1; end != NULL; end = end->next, i++)
{
if(i == r)
break;
}
for(temp1 = end->value, j = 1; temp1 != NULL; temp1 = temp1->next, j++)
{
```

```
if(j == c)
break;
}
if( i != r || j != c )
{
printf("\nGiven index is not valid !\n");
exit(0);
}
temp1->value = value;
}
int getValue(pArray *start, int r, int c)
{
pArray *end;
array *temp1;
int i, j;
for(end = start, i = 1; end != NULL; end = end->next, i++)
{
if(i == r)
break;
}
for(temp1 = end->value, j = 1; temp1 != NULL; temp1 = temp1->next, j++)
{
if(j == c)
break;
}
if( i != r || j != c )
{
printf("\nGiven index is not valid !\n");
exit(0);
}
return(temp1->value);
}
void display(pArray *start)
{
pArray *end;
array *temp1;
printf("\nArray =>\n");
for(end = start; end != NULL; end = end->next)
{
for(temp1 = end->value; temp1 != NULL; temp1 = temp1->next)
{
printf("%d ", temp1->value);
}
printf("\n");
}
}
void main()
{
pArray *pa, *ca;
int r1,r2,c1,c2;
int i,j,k;
printf("Enter numbers of Row and Column - ");
scanf(" %d %d", &r1, &c1);
createArray(&pa, r1, c1);
fillArray(pa);
display(pa);
r2 = 3;
```

```
c2 = 0;
for(i = 1; i<= r1; i++)
{
for(j=1;j<=c1;j++)
{
if(0 < getValue(pa, i, j))
{
c2++;
}
}
}
createArray(&ca, r2, c2);
display(ca);
k = 0;
for(i = 1; i<= r1; i++)
{
for(j=1;j<=c1;j++)
{
if(0 < getValue(pa, i, j))
{
if(k>=c2)
{
printf("\nError Occured ! Program terminated !\n");
exit(0);
}
k++;
putValue(ca, 1, k, i);
putValue(ca, 2, k, j);
putValue(ca, 3, k, getValue(pa, i, j));
}
}
}
display(pa);
display(ca);
                                                                                    }
```

```
// Problem Statement - insert element in to an array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
void main()
{
int array[200], index = -1;
int i;
while(1)
{
printf("\n1. Insert a new element in to array\n2. update a element\n3. display array
\n4. exit()\nEnter Your choice - ");
scanf(" %d", &i);
```

```
switch(i)
{
case 1: if(index < 200)
{
index++;
printf("\nEnter value for ne element - ");
scanf(" %d", &array[index]);
} else
{
printf("\narray is full !");
}
break;
case 2: printf("\nEnter value of index to update that element (start
count from 1 noy from 0) - ");
scanf(" %d", &i);
i = i - 1;
if(i <= index && i > -1)
{
printf("\nEnter value to update this imdex's
element - ");
scanf(" %d", &array[i]);
} else
{
printf("\nindex is outof range !\n");
}
break;
case 3: printf("\nArray =>> ");
if(index == -1)
printf("Nothig to show !\n");
else
{
for(i = 0; i<= index; i++)
printf("%d ", array[i]);
printf("\n");
}
break;
case 4: return;
default : printf("\Enter proper number from given option !\n");
}
}
}
// Problem Statement - Delete element form an array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
void main()
{
int array[50], index = 49;
int i,j;
for(i = 0; i<50; i++)
{
array[i] = i+1;
}
printf("\nThere is a array of 50 numbers .....\n");
printf("\nArray =>> ");
for(i = 0; i< 50; i++)
printf("%d ", array[i]);
```

```
printf("\n");
while(1)
{
printf("\n1. delete a element in to array\n2. update a element\n3. display
array\n4. exit()\nEnter Your choice - ");
scanf(" %d", &i);
switch(i)
{
case 1: if(index < 0)
{
printf("\nArray is empty !\n");
break;
}
printf("\nEnter value of index to delete that
element (start count from 1 noy from 0) - ");
scanf(" %d", &i);
i = i - 1;
if(i <= index && i > -1)
{
for(j = 0; j != i; j++);
while(j != index)
{
j++;
array[j-1] = array[j];
} index--;
} else
{
printf("\nindex is outof range !\n");
}
break;
case 2: printf("\nEnter value of index to update that element (start
count from 1 noy from 0) - ");
scanf(" %d", &i);
i = i - 1;
if(i <= index && i > -1)
{
printf("\nEnter value to update this imdex's
element - ");
scanf(" %d", &array[i]);
} else
{
printf("\nindex is outof range !\n");
}
break;
case 3: printf("\nArray =>> ");
if(index == -1)
printf("Nothig to show !\n");
else
{
for(i = 0; i <= index; i++)
printf("%d ", array[i]);
printf("\n");
}
break;
case 4: return;
default : printf("\Enter proper number from given option !\n");
}
```

```
}
}
// Problem Statement - merge two sorted array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
int *array1 = NULL, *array2 = NULL, *mergeArray = NULL;
void display(int *array);
void gets();
void display(int *array)
{
int i;
for(i =1; i<array[0]; i++)
{
printf("%d ", array[i]);
}
printf("\n");
}
void gets()
{
char x;
scanf("%c", &x);
while(x != '\n')
scanf("%c", &x);
}
void main()
{
int length;
int i;
printf("\nEnter length of first Array - ");
scanf(" %d", &length);
if(length<1)
{
printf("\nValue less than 1 is not valid ! Program terminated\n");
return;
} array1 =
(
int *)malloc((length +
1) *
sizeof(int));
if(array1 == NULL)
{
printf("\nError occured during memory allcation ...\nProgram terminated !\n");
return;
} array1[0] =
length+1;
printf("\nFill this first array with values in sorted order \n");
for(length = 1; length< array1[0]; length++)
{
scanf(" %d", &array1[length]);
}
gets();
printf("\nEnter length of second Array - ");
scanf(" %d", &length);
if(length<1)
{
printf("\nValue less than 1 is not valid ! Program terminated\n");
```

```
return;
} array2 =
(
int *)malloc((length +
1) *
sizeof(int));
if(array2 == NULL)
{
printf("\nError occured during memory allcation ...\nProgram terminated !\n");
return;
} array2[0] =
length+1;
printf("\nFill this second array with values in sorted order(sorting type must be same as
first array (both must be assending or decending))\n");
for(length = 1; length< array2[0]; length++)
{
scanf(" %d", &array2[length]);
}
gets();
mergeArray = (int *)malloc((array1[0] + array2[0] - 1) * sizeof(int));
if(mergeArray == NULL)
{
printf("\nError occured during memory allcation ...\nProgram
terminated !\n");
return;
}
mergeArray[0] = (array1[0] + array2[0] - 1);
length = 1;
i = 1;
for(; length < array1[0]; length++)
{
while(array1[length]>array2[i] && i < array2[0])
{
mergeArray[length + i -1] = array2[i];
i++;
}
mergeArray[length + i -1] = array1[length];
}
while( i < array2[0])
{
mergeArray[length + i -1] = array2[i];
i++;
}
printf("\nFirst Array => ");
display(array1);
printf("\nSecond Array => ");
display(array2);
printf("\nMerge Array => ");
display(mergeArray);
}
```



```
// Problem Statement - Linear search using array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
void gets()
{
char x;
```

```
scanf("%c", &x);
while(x != '\n')
scanf("%c", &x);
}
void main()
{
int *array, index, length;
int i;
char ans;
printf("\nHow many terms would be in the array - ");
scanf(" %d", &length);
gets();
if(length < 1)
{
printf("\nValue less than 1 is not valid ! Program Terminated !\n");
return;
} array =
(
int *)malloc(length *
sizeof(int));
if(array == NULL)
{
printf("\nError occured during memory allocation !. Program
terminated !\n");
return;
}
printf("\nFill the array with value .....\n");
for(index = 0; index<length; index++)
{
scanf("%d", &array[index]);
}
gets();
printf("\nComplete !\n\n");
printf("\nYour Array => ");
for(index=0; index<length; index++)
printf("%d ", array[index]);
loop:
printf("\n\nEnter a value to search if this value is present in array or not \nEnter your
value - ");
scanf(" %d", &i);
for(index = 0; index < length; index++)
{
if(array[index] == i)
{
break;
}
} if(index == length)
{
printf("\nThe value you are searching for is not present in this list !\n");
} else
{
printf("\nValue found in this list ! Index no is %d\n", index);
}
printf("\nDo You want to search another number ( Y or N ) ? -> ");
gets();
scanf("%c", &ans);
if(ans == 'Y' || ans == 'y')
```

```
goto loop;
printf("\nEnd of Program !\n");
}
// Problem Statement - Binary search using array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
int *array, arrayIndex, length;
void gets()
{
char x;
scanf("%c", &x);
while(x != '\n')
scanf("%c", &x);
}
int binarySearch(int value, int start, int end)
{
arrayIndex = (start + end)/2;
if(array[arrayIndex] == value)
{
printf("\nValue found ! index=> %d\n", arrayIndex);
return(1);
}
if(array[arrayIndex] > value)
{
if(array[0] < array[length-1])
end = arrayIndex-1;
else
start = arrayIndex+1;
} else
{
if(array[0] < array[length-1])
start = arrayIndex+1;
else
end = arrayIndex-1;
}
return(binarySearch(value, start, end));
}
void main()
{
int i;
char ans;
printf("\nHow many terms would be in the array - ");
scanf(" %d", &length);
gets();
if(length < 1)
{
printf("\nValue less than 1 is not valid ! Program Terminated !\n");
return;
} array =
(
int *)malloc(length *
sizeof(int));
if(array == NULL)
{
printf("\nError occured during memory allocation !. Program
terminated !\n");
```

```
return;
}
printf("\nFill the array with value in sorted order .....\n");
for(arrayIndex = 0; arrayIndex<length; arrayIndex++)
{
scanf("%d", &array[arrayIndex]);
}
gets();
printf("\nComplete !\n\n");
printf("\nYour Array => ");
for(arrayIndex=0; arrayIndex<length; arrayIndex++)
printf("%d ", array[arrayIndex]);
loop:
printf("\n\nEnter a value to search if this value is present in array or not \nEnter your
value - ");
scanf(" %d", &i);
if(!binarySearch(i, 0, length-1))
{
printf("\nThe value you are searching for is not present in this list !\n");
}
printf("\nDo You want to search another number ( Y or N ) ? -> ");
gets();
scanf("%c", &ans);
if(ans == 'Y' || ans == 'y')
goto loop;
printf("\nEnd of Program !\n");
}
// Problem Statement - Interpolation search using array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
int *array, arrayIndex, length;
void gets()
{
char x;
scanf("%c", &x);
while(x != '\n')
scanf("%c", &x);
}
int interpolationSearch(int value, int start, int end)
{
arrayIndex = start + ((end - start) /(float) (array[end] - array[start]) * (value -
array[start]));
if(arrayIndex >= length)
{
return 0;
} if(array[arrayIndex] == value)
{
printf("\nValue found ! index=> %d\n", arrayIndex);
return(1);
}
if(array[arrayIndex] > value)
{
if(array[0] < array[length-1])
end = arrayIndex-1;
else
start = arrayIndex+1;
```

```
return(interpolationSearch(value, start, end));
} else
{
if(array[0] < array[length-1])
start = arrayIndex+1;
else
end = arrayIndex-1;
return(interpolationSearch(value, start, end));
}
return 0;
}
void main()
{
int i;
char ans;
printf("\nHow many terms would be in the array - ");
scanf(" %d", &length);
gets();
if(length < 1)
{
printf("\nValue less than 1 is not valid ! Program Terminated !\n");
return;
} array =
(
int *)malloc(length *
sizeof(int));
if(array == NULL)
{
printf("\nError occured during memory allocation !. Program
terminated !\n");
return;
}
printf("\nFill the array with value in sorted and distributed manner .....\
n");
for(arrayIndex = 0; arrayIndex<length; arrayIndex++)
{
scanf("%d", &array[arrayIndex]);
}
gets();
printf("\nComplete !\n\n");
printf("\nYour Array => ");
for(arrayIndex=0; arrayIndex<length; arrayIndex++)
printf("%d ", array[arrayIndex]);
loop:
printf("\n\nEnter a value to search if this value is present in array or not \nEnter
your value - ");
scanf(" %d", &i);
if(!interpolationSearch(i, 0, length-1))
{
printf("\nThe value you are searching for is not present in this list !\n");
}
printf("\nDo You want to search another number ( Y or N ) ? -> ");
gets();
scanf("%c", &ans);
gets();
if(ans == 'Y' || ans == 'y')
goto loop;
```

```
printf("\nEnd of Program !\n");
}
//problem statement – Polynomial implementation using Array
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
void main()
{
int p1[20][2], p2[20][2], sum[40][2];
int l1,l2,l3;
int i,j;
printf("Both of two polynomial will be function of x ( f(x) )\n");
printf("\nEnter number of terms in first polynomial (number of terms mustbe less than
or equal to 20) - ");
scanf(" %d", &l1);
printf("\nEnter data in first polynomial -\n");
for(i=0; i<l1; i++)
{
printf("for term no %d\n", i+1);
printf("coefficient -> ");
scanf(" %d", &p1[i][0]);
printf("power -> ");
scanf(" %d", &p1[i][1]);
}
printf("\n");
printf("\nEnter number of terms in second polynomial (number of terms mustbe less
than
or equal to 20) - ");
scanf(" %d", &l2);
printf("\nEnter data in second polynomial -\n");
for(i=0; i<l2; i++)
{
printf("for term no %d\n", i+1);
printf("coefficient -> ");
scanf(" %d", &p2[i][0]);
printf("power -> ");
scanf(" %d", &p2[i][1]);
}
printf("\n");
l3=-1;
for(i=0;i<l1;i++)
{
l3++;
sum[l3][0] = p1[i][0];
sum[l3][1] = p1[i][1];
}
for(i=0;i<l2;i++)
{
for(j=0;j<=l3;j++)
{
if(p2[i][1] == sum[j][1])
{
sum[j][0] = sum[j][0] + p2[i][0];
break;
}
} if(j>l3)
{
l3++;
```

```
sum[l3][0] = p2[i][0];
sum[l3][1] = p2[i][1];
}
}
printf("\nResults =>>\n");
printf("Polynomial 1 =>> ");
for(i=0;i<l1;i++)
{
if(p1[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", p1[i][0], 'x', p1[i][1]);
}
printf("\n");
printf("Polynomial 2 =>> ");
for(i=0;i<l2;i++)
{
if(p2[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", p2[i][0], 'x', p2[i][1]);
}
printf("\n");
printf("Sum of Polynomial 1 and 2 =>> ");
for(i=0;i<=l3;i++)
{
if(sum[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", sum[i][0], 'x', sum[i][1]);
}
printf("\n");
}
//problem statement - multiplication of two polynomial
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
void main()
{
int p1[20][2], p2[20][2], mul[400][2];
int l1,l2,l3;
int i,j,k;
int v;
printf("Both of two polynomial will be function of x ( f(x) )\n");
printf("\nEnter number of terms in first polynomial (number of terms mustbe less than
or equal to 20) - ");
scanf(" %d", &l1);
printf("\nEnter data in first polynomial -\n");
for(i=0; i<l1; i++)
{
printf("for term no %d\n", i+1);
printf("coefficient -> ");
scanf(" %d", &p1[i][0]);
printf("power -> ");
scanf(" %d", &p1[i][1]);
}
printf("\n");
printf("\nEnter number of terms in second polynomial (number of terms mustbe
less than or equal to 20) - ");
scanf(" %d", &l2);
printf("\nEnter data in second polynomial -\n");
```

```
for(i=0; i<l2; i++)
{
printf("for term no %d\n", i+1);
printf("coefficient -> ");
scanf(" %d", &p2[i][0]);
printf("power -> ");
scanf(" %d", &p2[i][1]);
}
printf("\n");
l3=0;
for(i=0;i<l1;i++)
{
for(j=0;j<l2;j++)
{
v = p1[i][1] + p2[j][1];
for(k=0;k<l3;k++)
{
if(v == mul[k][1])
{
mul[k][0] = mul[k][0] + p1[i][0] * p2[j][0];
break;
}
} if(!(k<l3))
{
mul[k][0] = p1[i][0] * p2[j][0];
mul[k][1] = p1[i][1] + p2[j][1];
l3++;
}
}
}
printf("\nResults =>>\n");
printf("Polynomial 1 =>> ");
for(i=0;i<l1;i++)
{
if(p1[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", p1[i][0], 'x', p1[i][1]);
}
printf("\n");
printf("Polynomial 2 =>> ");
for(i=0;i<l2;i++)
{
if(p2[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", p2[i][0], 'x', p2[i][1]);
}
printf("\n");
printf("Multiplication of Polynomial 1 and 2 =>> ");
for(i=0;i<l3;i++)
{
if(mul[i][0] > -1)
printf("+");
printf("%d%c^(%d) ", mul[i][0], 'x', mul[i][1]);
}
printf("\n");
}
//problem statement - Sparse matrix to Compact matrix
```

```
// gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
#include<stdio.h>
#include<malloc.h>
void main()
{
int **p, **q;
int r,c;
int i,j,k,count;
printf("\nEnter row and column number of matrix - ");
scanf(" %d %d", &r, &c);
p = (int **)malloc(c*sizeof(int*));
if(p == NULL)
{
printf("\nError occured during memory allocation ! Program
terminated !\n");
return;
}
for(i =0; i<c;i++)
{
p[i] = (int *)malloc(r*sizeof(int));
if(p[i] == NULL)
{
printf("\nError occured during memory allocation ! Program
terminated !\n");
return;
}
}
printf("\nFill the matrix with value \n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf(" %d", &p[i][j]);
}
}
printf("\n");
k = 0;
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
if(p[i][j] != 0)
{
k++;
}
}
}
q = (int **)malloc(k*sizeof(int*));
if(q == NULL)
{
printf("\nError occured during memory allocation ! Program
terminated !\n");
return;
}
for(i =0; i< k;i++)
{
q[i] = (int *)malloc(3*sizeof(int));
```

```
if(q[i] == NULL)
{
printf("\nError occured during memory allocation ! Program
terminated !\n");
return;
}
} count =
0;
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
if(p[i][j] != 0)
{
q[count][0] = i;
q[count][1] = j;
q[count][2] = p[i][j];
count++;
}
}
}
printf("\nSparse Matrix =>\n");
for(i = 0; i < r; i++)
{
for(j =0; j<c; j++)
printf(" %d", p[i][j]);
printf("\n");
}
printf("\n");
printf("\nCompact Matrix =>\n");
for(i = 0; i < 3; i++)
{
for(j =0; j<k; j++)
printf(" %d", q[j][i]);
printf("\n");
}
printf("\n");
```

}

Shell sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define ELEMENT_NR 20
#define ARRAY_LEN(x) (sizeof(x) / sizeof((x)[0]))
const char *notation = "Shell Sort Big O Notation:\
                                                    \n--> Best Case: O(n log(n)) \
                                                    \n--> Average Case: depends on
gap sequence \
                                                    \n--> Worst Case: O(n)";

void show_data(int arr[], int len)
{
    int i;

    for (i = 0; i < len; i++)
```

```
        printf("%3d ", arr[i]);
    printf("\n");
}

void swap(int *a, int *b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
}

void shellSort(int array[], int len)
{
    int i, j, gap;

    for (gap = len / 2; gap > 0; gap = gap / 2)
        for (i = gap; i < len; i++)
            for (j = i - gap; j >= 0 && array[j] > array[j + gap]; j = j -
gap)
                swap(&array[j], &array[j + gap]);
}

int main(int argc, char *argv[])
{
    int i;
    int array[ELEMENT_NR];
    int range = 500;
    int size;
    clock_t start, end;
    double time_spent;

    srand(time(NULL));
    for (i = 0; i < ELEMENT_NR; i++)
        array[i] = rand() % range + 1;

    size = ARRAY_LEN(array);

    show_data(array, size);
    start = clock();
    shellSort(array, size);
    end = clock();
    time_spent = (double)(end - start) / CLOCKS_PER_SEC;

    printf("Data Sorted\n");
    show_data(array, size);

    printf("%s\n", notation);
    printf("Time spent sorting: %f\n", time_spent);

    return 0;
}
```

Radix sort

```
//sorting of array list using Radix sort
#include <stdio.h>

#define range 10  // Range for integers is 10 as digits range from 0-9

// Utility function to get the maximum value in ar[]
int MAX(int ar[], int size){
    int i, max = ar[0];
    for(i = 0; i<size; i++){
        if(ar[i]>max)
            max = ar[i];
    }
    return max;
```

```
}

// Counting sort according to the digit represented by place
void countSort(int arr[],int n,int place)
{
        int i,freq[range]={0};
        int output[n];

        // Store count of occurences in freq[]
        for(i=0;i<n;i++)
                freq[(arr[i]/place)%range]++;

        // Change freq[i] so that it contains the actual position of the
digit in output[]
        for(i=1;i<range;i++)
                freq[i]+=freq[i-1];

        // Build the output array
        for(i=n-1;i>=0;i--)
        {
                output[freq[(arr[i]/place)%range]-1]=arr[i];
                freq[(arr[i]/place)%range]--;
        }

        // Copy the output array to arr[], so it contains numbers according
to the current digit
        for(i=0;i<n;i++)
                arr[i]=output[i];
}

/*This is where the sorting of the array takes place
 arr[] --- Array to be sorted
 n --- Array Size
 max --- Maximum element in Array
 */
void radixsort(int arr[],int n,int max)            //max is the maximum
element in the array
{
        int mul=1;
        while(max)
        {
                countsort(arr,n,mul);
                mul*=10;
                max/=10;
        }
}

int main(int argc, const char * argv[]){
    int n;
    printf("Enter size of array:\n");
    scanf("%d", &n); // E.g. 8

    printf("Enter the elements of the array\n");
    int i;
    int arr[n];
    for(i = 0; i < n; i++){
        scanf("%d", &arr[i] );
    }

    printf("Original array: ");
    display(arr, n);                // Original array : 10 11 9 8 4 7 3 8

    int max;
    max = MAX(arr,n);

    radixsort(arr, n, max);

    printf("Sorted array: ");
    display(arr, n);                // Sorted array : 3 4 7 8 8 9 10 11
```

```
    return 0;


}
```

Merge sort

```
#include <stdio.h>

void swap (int *a,int *b)//To swap the variables//
{
    int t;
    t= *a;
    *a=*b;
    *b=t;

}

void merge(int a[],int l,int r,int n)//To merge //
{  int *b = (int*)malloc(n*sizeof(int));
int c=l;
    int p1,p2;
    p1 = l;p2=((l+r)/2)+1;
    while ((p1<((l+r)/2)+1) &&(p2<r+1))
    { if(a[p1] <= a[p2])
{          b[c++] = a[p1];
           p1++;
}
          else
          {b[c++] = a[p2];p2++;}

    };

    if (p2 == r+1)
    {
       while ((p1<((l+r)/2)+1))
    {
          b[c++] = a[p1];
            p1++;
    };

    }
    else
    {
          while ((p2<r+1))
    {
          b[c++] = a[p2];
p2++;
    };
         }

        for (c=l;c<r-l+1;c++)
        a[c] = b[c];

}
void mergesort(int a[],int n,int l,int r)
{
    if (r-l == 1 )
    {
        if (a[l]>a[r])
        swap(&a[l],&a[r]);

    }
    else if(l==r)
    {}
    else
    {mergesort(a,n,l,(l+r)/2);
     mergesort(a,n,((l+r)/2)+1,r);
```

```
       merge(a,l,r,n);

    }

}
int main(void) {   //main function//
int *a,n,i;
scanf("%d",&n);
a = (int*)malloc(n*sizeof(int));
for (i=0;i<n;i++)
{

    scanf("%d",&a[i]);

}

mergesort(a,n,0,n-1);
for (i=0;i<n;i++)
{

    printf(" %d",&a[i]);

}


        return 0;
}
```

Insertion sort

```
//sorting of array list using insertion sort
#include <stdio.h>

/*Displays the array, passed to this method*/
void display(int arr[], int n) {
    int i;
    for(i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}

/*This is where the sorting of the array takes place
 arr[] --- Array to be sorted
 size --- Array Size
 */
void insertionSort(int arr[], int size) {
    int i, j, key;
    for(i = 0; i < size; i++) {
        j = i - 1;
        key = arr[i];
        /* Move all elements greater than key to one position */
        while(j >= 0 && key < arr[j]) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        /* Find a correct position for key */
        arr[j + 1] = key;
    }
}

int main(int argc, const char * argv[]) {
    int n;
    printf("Enter size of array:\n");
    scanf("%d", &n); // E.g. 8
```

```
    printf("Enter the elements of the array\n");
    int i;
    int arr[n];
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i] );
    }

    printf("Original array: ");
    display(arr, n);

    insertionSort(arr, n);

    printf("Sorted array: ");
    display(arr, n);

    return 0;
}
```

Selection sort

```
//sorting of array list using selection sort
#include <stdio.h>

/*Displays the array, passed to this method*/
void display(int arr[], int n){

    int i;
    for(i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }

    printf("\n");

}

/*Swap function to swap two values*/
void swap(int *first, int *second){

    int temp = *first;
    *first = *second;
    *second = temp;

}

/*This is where the sorting of the array takes place
 arr[] --- Array to be sorted
 size --- Array Size
 */
void selectionSort(int arr[], int size){

    for(int i=0; i<size; i++) {
        int min_index = i;
        for(int j=i+1; j<size; j++) {
            if(arr[min_index] > arr[j]) {
                min_index = j;
            }
        }
        swap(&arr[i], &arr[min_index]);
    }
}
```



```
int main(int argc, const char * argv[]) {
    int n;
    printf("Enter size of array:\n");
    scanf("%d", &n); // E.g. 8

    printf("Enter the elements of the array\n");
    int i;
    int arr[n];
```

```
    for(i = 0; i < n; i++){
        scanf("%d", &arr[i] );
    }

    printf("Original array: ");
    display(arr, n);                    // Original array : 10 11 9 8 4 7 3 8

    selectionSort(arr, n);

    printf("Sorted array: ");
    display(arr, n);                // Sorted array : 3 4 7 8 8 9 10 11

    return 0;
}
```

Quick sort

```
#include <stdio.h>
#include <stdlib.h>

/*Displays the array, passed to this method*/
void display(int arr[], int n)
{

    int i;
    for (i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }

    printf("\n");
}

/*Swap function to swap two values*/
void swap(int *first, int *second)
{

    int temp = *first;
    *first = *second;
    *second = temp;
}

/*Partition method which selects a pivot
  and places each element which is less than the pivot value to its left
  and the elements greater than the pivot value to its right
  arr[] --- array to be partitioned
  lower --- lower index
  upper --- upper index
*/
int partition(int arr[], int lower, int upper)
{

    int i = (lower - 1);

    int pivot = arr[upper]; // Selects last element as the pivot value

    int j;
    for (j = lower; j < upper; j++)
    {
        if (arr[j] <= pivot)
        { // if current element is smaller than the pivot

            i++; // increment the index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
```

```
    swap(&arr[i + 1], &arr[upper]); // places the last element i.e, the pivot
to its correct position

    return (i + 1);
}

/*This is where the sorting of the array takes place
        arr[] --- Array to be sorted
        lower --- Starting index
        upper --- Ending index
*/
void quickSort(int arr[], int lower, int upper)
{

    if (upper > lower)
    {

        // partitioning index is returned by the partition method , partition
element is at its correct poition

        int partitionIndex = partition(arr, lower, upper);

        // Sorting elements before and after the partition index
        quickSort(arr, lower, partitionIndex - 1);
        quickSort(arr, partitionIndex + 1, upper);
    }
}

int main()
{

    int n;
    printf("Enter size of array:\n");
    scanf("%d", &n); // E.g. 8

    printf("Enter the elements of the array\n");
    int i;
    int *arr = (int *)malloc(sizeof(int) * n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    printf("Original array: ");
    display(arr, n); // Original array : 10 11 9 8 4 7 3 8

    quickSort(arr, 0, n - 1);

    printf("Sorted array: ");
    display(arr, n); // Sorted array : 3 4 7 8 8 9 10 11
    getchar();
    return 0;
}
```

Heap sort

```
#include <stdio.h>

void heapify(int *unsorted, int index, int heap_size);
void heap_sort(int *unsorted, int n);
int main() {
        int n = 0;
        int i = 0;
        char oper;

        int* unsorted;
        printf("Enter the size of the array you want\n");
        scanf("%d", &n);
        unsorted = (int*)malloc(sizeof(int) * n);
```

```
        while (getchar() != '\n');
        printf("Enter numbers separated by a comma:\n");
        while (i != n) {
                scanf("%d,", (unsorted + i));
                i++;
        }
        heap_sort(unsorted, n);

        printf("[");
        printf("%d", *(unsorted));
        for (int i = 1; i < n; i++) {
                printf(", %d", *(unsorted + i));
        }
        printf("]");
}

void heapify(int *unsorted, int index, int heap_size) {
        int temp;
        int largest = index;
        int left_index = 2 * index;
        int right_index = 2 * index + 1;
        if (left_index < heap_size && *(unsorted + left_index) > *(unsorted +
largest)) {
                largest = left_index;
        }
        if (right_index < heap_size && *(unsorted + right_index) > *(unsorted
+ largest)) {
                largest = right_index;
        }

        if (largest != index) {
                temp = *(unsorted + largest);
                *(unsorted + largest) = *(unsorted + index);
                *(unsorted + index) = temp;
                heapify(unsorted, largest, heap_size);
        }
}

void heap_sort(int *unsorted, int n) {
        int temp;
        for (int i = n / 2 - 1; i > -1; i--) {
                heapify(unsorted, i, n);
        }
        for (int i = n - 1; i > 0; i--) {
                temp = *(unsorted);
                *(unsorted) = *(unsorted + i);
                *(unsorted + i) = temp;
                heapify(unsorted, 0, i);
        }
}
```

Bucket sort

```
/*
* Algorithm : Bucket Sort
* Time-Complexity : O(n)
*/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
```



```
#define NARRAY 8  /* array size */
#define NBUCKET 5 /* bucket size */
#define INTERVAL 10 /* bucket range */

struct Node
{
    int data;
    struct Node *next;
```

```
};

void BucketSort(int arr[]);
struct Node *InsertionSort(struct Node *list);
void print(int arr[]);
void printBuckets(struct Node *list);
int getBucketIndex(int value);

void BucketSort(int arr[])
{
    int i,j;
    struct Node **buckets;

    /* allocate memory for array of pointers to the buckets */
    buckets = (struct Node **)malloc(sizeof(struct Node*) * NBUCKET);

    /* initialize pointers to the buckets */
    for(i = 0; i < NBUCKET; ++i)
    {
        buckets[i] = NULL;
    }

    /* put items into the buckets */
    /* creates a link list in each bucket slot */
    for(i = 0; i < NARRAY; ++i)
    {
        struct Node *current;
        int pos = getBucketIndex(arr[i]);
        current = (struct Node *) malloc(sizeof(struct Node));
        current->data = arr[i];
        current->next = buckets[pos];
        buckets[pos] = current;
    }

    /* check what's in each bucket */
    for(i = 0; i < NBUCKET; i++)
    {
        printf("Bucket[\"%d\"] : ", i);
        printBuckets(buckets[i]);
        printf("\n");
    }

    /* sorting bucket using Insertion Sort */
    for(i = 0; i < NBUCKET; ++i)
    {
        buckets[i] = InsertionSort(buckets[i]);
    }

    /* check what's in each bucket */
    printf("--------------\n");
    printf("Buckets after sorted\n");
    for(i = 0; i < NBUCKET; i++)
    {
        printf("Bucket[\"%d\"] : ", i);
        printBuckets(buckets[i]);
        printf("\n");
    }

    /* put items back to original array */
    for(j =0, i = 0; i < NBUCKET; ++i)
```



```
        struct Node *node;
        node = buckets[i];
        while(node)
        {

            // precondition for avoiding out of bounds by the array
            assert(j < NARRAY);
            arr[j++] = node->data;
```

```
            node = node->next;
        }
    }

    /* free memory */
    for(i = 0; i < NBUCKET; ++i)
    {
        struct Node *node;
        node = buckets[i];
        while(node)
        {
            struct Node *tmp;
            tmp = node;
            node = node->next;
            free(tmp);
        }
    }
    free(buckets);
    return;
}

/* Insertion Sort */
struct Node *InsertionSort(struct Node *list)
{
    struct Node *k,*nodeList;
    /* need at least two items to sort */
    if(list == NULL || list->next == NULL)
    {
        return list;
    }

    nodeList = list;
    k = list->next;
    nodeList->next = NULL; /* 1st node is new list */
    while(k != NULL)
    {
        struct Node *ptr;
        /* check if insert before first */
        if(nodeList->data > k->data)
        {
            struct Node *tmp;
            tmp = k;
            k = k->next; // important for the while
            tmp->next = nodeList;
            nodeList = tmp;
            continue;
        }

        // from begin up to end
        // finds [i] > [i+1]
        for(ptr = nodeList; ptr->next != NULL; ptr = ptr->next)
        {
            if(ptr->next->data > k->data) break;
        }

        // if found (above)
        if(ptr->next != NULL)
        {
            struct Node *tmp;
            tmp = k;
            k = k->next; // important for the while
            tmp->next = ptr->next;
            ptr->next = tmp;
            continue;
        }
        else
        {
            ptr->next = k;
            k = k->next; // important for the while
```

```
            ptr->next->next = NULL;
            continue;
        }
    }
    return nodeList;
}

int getBucketIndex(int value)
{
    return value/INTERVAL;
}

void print(int ar[])
{
    int i;
    for(i = 0; i < NARRAY; ++i)
    {
        printf("%d ", ar[i]);
    }
    printf("\n");
}

void printBuckets(struct Node *list)
{
    struct Node *cur = list;
    while(cur)
    {
        printf("%d ", cur->data);
        cur = cur->next;
    }
}

int main(void)
{
    int array[NARRAY] = {29,25,-1,49,9,37,21,43};

    printf("Initial array\n");
    print(array);
    printf("------------\n");

    BucketSort(array);
    printf("------------\n");
    printf("Sorted array\n");
    print(array);
    return 0;
}
```

Bubble sort

```
//sorting of array list using bubble sort
#include <stdio.h>

/*Displays the array, passed to this method*/
void display(int arr[], int n){

    int i;
    for(i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");

}

/*Swap function to swap two values*/
void swap(int *first, int *second){

    int temp = *first;
    *first = *second;
```

```
    *second = temp;

}

/*This is where the sorting of the array takes place
 arr[] --- Array to be sorted
 size --- Array Size
 */
void bubbleSort(int arr[], int size){

    for(int i=0; i<size-1; i++) {
        for(int j=0; j<size-1-i; j++) {
            if(arr[j]>arr[j+1]) {
                swap(&arr[j], &arr[j+1]);
            }
        }
    }
}

int main(int argc, const char * argv[]) {
    int n;
    printf("Enter size of array:\n");
    scanf("%d", &n); // E.g. 8

    printf("Enter the elements of the array\n");
    int i;
    int arr[n];
    for(i = 0; i < n; i++){
        scanf("%d", &arr[i] );
    }

    printf("Original array: ");
    display(arr, n);                    // Original array : 10 11 9 8 4 7 3 8

    bubbleSort(arr, n);

    printf("Sorted array: ");
    display(arr, n);                // Sorted array : 3 4 7 8 8 9 10 11

    return 0;
}
```

Interpolation search

```
#include<stdio.h>

/* By comparison, binary search always chooses the middle of the remaining
 * search space, discarding one half or the other, depending on the
comparison
 * between the key found at the estimated position and the key sought. The
remaining
 * search space is reduced to the part before or after the estimated
position.
 * The linear search uses equality only as it compares elements one-by-one
from the start, ignoring any sorting.
 * On average the interpolation search makes about log(log(n)) comparisons
(if the elements
 * are uniformly distributed), where n is the number of elements to be
searched. In the worst case
 * (for instance where the numerical values of the keys increase
exponentially) it can make up to O(n) comparisons.
 * In interpolation-sequential search, interpolation is used to find an item
near the one being searched for,
 * then linear search is used to find the exact item. */

int interpolationSearch(int arr[], int n, int key)
```

```
{
    int low = 0, high = n - 1;
    while (low <= high && key >= arr[low] && key <= arr[high]) {
        /* Calculate the nearest posible position of key */
        int pos = low + ((key - arr[low]) * (high - low)) / (arr[high] -
arr[low]);
        if (key > arr[pos])
            low = pos + 1;
        else if (key < arr[pos])
            high = pos - 1;
        else /* Found */
            return pos;
    }
    /* Not found */
    return -1;
}


int main()
{
    int x;
    int arr[] =  {10, 12, 13, 16, 18, 19, 20, 21, 22, 23,
                  24, 33, 35, 42, 47};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Array: ");
    for(int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\nEnter the number to be searched: ");
    scanf("%d",&x); /* Element to be searched */

    int index = interpolationSearch(arr, n, x);

    /* If element was found */
    if (index != -1)
        printf("Element found at position: %d\n", index);
    else
        printf("Element not found.\n");
    return 0;
}
```

Binary search

```
#include <stdio.h>

// Recursive Function- It returns location of x assumiung array arr[l..r] is
present, otherwise -1

int binarysearch(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
        int mid = l + (r - l)/2;

        // If element is present at middle
        if (arr[mid] == x)  return mid;

        // If element is smaller than middle
        if (arr[mid] > x) return binarysearch(arr, l, mid-1, x);

        // Else element is in right subarray
        return binarysearch(arr, mid+1, r, x);
   }

   // When element is not present in array
   return -1;
}

int main(void)
```

```
{
   // give function an array to work with
   int arr[] = {2, 3, 4, 10, 40};
   // get size of array
   int n = sizeof(arr)/ sizeof(arr[0]);
   //set value to look for
   int x = 10;
   // set result to what is returned from binarysearch
   int result = binarysearch(arr, 0, n-1, x);
   // print out result
   (result == -1) ? printf("Element is not in the array\n")
                  : printf("Element is present at index %d\n", result);
   return 0;
}
```

Linear search

```
#include <stdio.h>

int linearsearch(int *arr, int size, int val){
        int i;
        for (i = 0; i < size; i++){
                if (arr[i] == val)
                        return 1;
        }
        return 0;
}

void main(){
        int n,i,v;
        printf("Enter the size of the array:\n");
        scanf("%d",&n); //Taking input for the size of Array

        int a[n];
        printf("Enter the contents for an array of size %d:\n", n);
        for (i = 0; i < n; i++)        scanf("%d", &a[i]);// accepts the
values of array elements until the loop terminates//

        printf("Enter the value to be searched:\n");
        scanf("%d", &v); //Taking input the value to be searched
        if (linearsearch(a,n,v))
                printf("Value %d is in the array.\n", v);
        else
                printf("Value %d is not in the array.\n", v);
}
```

Stack

```
/**
 * Kyler Smith, 2017
 * Stack data structure implementation.
 */

////////////////////////////////////////////////////////////////////////////
///
//INCLUDES
#include <stdio.h>
#include <stdlib.h>
```



```
////////////////////////////////////////////////////////////////////////////
///
//MACROS: CONSTANTS

////////////////////////////////////////////////////////////////////////////
///
//DATA STRUCTURES
struct node {
    int data;
```

```
    struct node* next;
    struct node* pre;
} *head, *tmp;

////////////////////////////////////////////////////////////////////////////////
///
//GLOBAL VARIABLES
int count = 0;

////////////////////////////////////////////////////////////////////////////////
///
//FUNCTION PROTOTYPES
void create();
void push(int x);
int pop();
int peek();
int size();
int isEmpty();

////////////////////////////////////////////////////////////////////////////////
///
//MAIN ENTRY POINT

int main(int argc, char const *argv[]) {

    int x, y, z;

    create();
    push(4);
    x = pop();
    // 4. Count: 0. Empty: 1.
    printf("%d.\t\tCount: %d.\tEmpty: %d.\n", x, size(), isEmpty());

    push(1);
    push(2);
    push(3);
    x = pop();
    y = pop();
    // 3, 2. Count: 1. Empty: 0;
    printf("%d, %d.\t\tCount: %d.\tEmpty: %d.\n", x, y, size(), isEmpty());
    pop();  // Empty the stack.

    push(5);
    push(6);
    x = peek();
    push(7);
    y = pop();
    push(8);
    z = pop();
    // 1, 6, 7, 8. Count: 2. Empty: 0.
    printf("%d, %d, %d.\tCount: %d.\tEmpty: %d.\n", x, y, z, size(),
isEmpty());

        return 0;
}

/**
 * Initialize the stack to NULL.
 */
void create() {
    head = NULL;
}

/**
 * Push data onto the stack.
 */
void push(int x) {
    if(head == NULL) {
        head = (struct node *)malloc(1 * sizeof(struct node));
```

```
        head->next = NULL;
        head->pre = NULL;
        head->data = x;
    } else {
        tmp = (struct node *)malloc(1 * sizeof(struct node));
        tmp->data = x;
        tmp->next = NULL;
        tmp->pre = head;
        head->next = tmp;
        head = tmp;
    }
    ++count;
}

/**
 * Pop data from the stack
 */
int pop() {
    int returnData;
    if(head == NULL) {
        printf("ERROR: Pop from empty stack.\n");
        exit(1);
    } else {
        returnData = head->data;

        if(head->pre == NULL){
            free(head);
            head = NULL;
        }
        else {
            head = head->pre;
            free(head->next);
        }
    }
    --count;
    return returnData;
}

/**
 * Returns the next value to be popped.
 */
int peek() {
    if(head != NULL)
        return head->data;
    else {
        printf("ERROR: Peeking from empty stack.");
        exit(1);
    }
}

/**
 * Returns the size of the stack.
 */
int size() {
    return count;
}

/**
 * Returns 1 if stack is empty, returns 0 if not empty.
 */
int isEmpty() {
    if(count == 0)
        return 1;
    return 0;
}
```


Queue

```
////////////////////////////////////////////////////////////////////////////
///
//INCLUDES
#include <stdio.h>
#include <stdlib.h>

////////////////////////////////////////////////////////////////////////////
///
//MACROS: CONSTANTS

////////////////////////////////////////////////////////////////////////////
///
//DATA STRUCTURES
struct node {
    int data;
    struct node* next;
    struct node* pre;
} *head, *tail, *tmp;

////////////////////////////////////////////////////////////////////////////
///
//GLOBAL VARIABLES
int count;

////////////////////////////////////////////////////////////////////////////
///
//FORWARD DECLARATIONS
void create();
void enque(int x);
int deque();
int peek();
int size();
int isEmpty();

////////////////////////////////////////////////////////////////////////////
///
//MAIN ENTRY POINT

int main(int argc, char const *argv[]) {

    create();
    enque(5);


        return 0;
}


void create() {
    head = NULL;
    tail = NULL;
}

/**
 * Puts an item into the Queue.
 */
void enque(int x) {
    if(head == NULL) {
        head = (struct node *)malloc(1 * sizeof(struct node));
        head->data = x;
        head->next = NULL;
        tail = head;
    } else {
        tmp = (struct node *)malloc(1 * sizeof(struct node));
        tmp->data = x;
        tmp->next = tail;
        tail = tmp;
    }
}
```

```
/**
 * Takes the next item from the Queue.
 */
int deque() {
    int returnData;
    if(head == NULL) {
        printf("ERROR: Deque from empty queue.\n");
        exit(1);
    } else {
        returnData = head->data;
        if(head->pre == NULL)
            head = NULL;
        else
            head = head->pre;
        head->next = NULL;
    }
}

/**
 * Returns the size of the Queue.
 */
int size() {
    return count;
}
```


doublyEndedQueue-array.c

```
#include<stdio.h>
#include<stdlib.h>


int size, front = -1, rear = -1, dequeue[10], item;
```

```
void insertQueueRear()
{
        if (rear == size-1)
        {
                printf("\n QUEUE OVERRFLOW !!!");
        }
        else
        {
                if (front == -1)
                {
                        front = 0;
                }
                printf("\n Enter the element to be inserted : ");
                scanf("%d", &item);
                rear++;
                dequeue[rear] = item;
        }
}


void insertQueueFront()
{
        if (front == 0)
        {
                printf("\n QUEUE OVERFLOW !!!");
        }
        else
        {
                if (front == -1)
                {
                        front = rear = 0;
                }
                else
                {
                        front = front -1;
                }

                printf("\n Enter the element to be inserted : ");
                scanf("%d", &item);
                dequeue[front] = item;
        }
}


void deleteQueueRear()
{
        if (rear == -1)
        {
                printf("\n QUEUE UNDERFLOW !!!");
        }
        else
        {
                printf("\n Element %d has been deleted from the rear",
dequeue[rear]);
                rear--;
        }
}
```



```
void deleteQueueFront()
{
        if (front == -1 || front > rear)
        {
                printf("\n QUEUE UNDERFLOW !!!");
        }
        else
        {
```

```
                printf("\n Element %d has been deleted from the front",
dequeue[front]);
                front++;
                if (front > rear)
                {
                        front = rear = -1;
                }
        }
}


void display()
{
        int i;

        if (front == -1)
        {
                printf("\n List is empty !!!");
        }
        else
        {
                printf("\n");
                for (i = front; i <= rear; ++i)
                {
                        printf("  %d <-", dequeue[i]);
                }
        }
        printf("\n");
}


void main()
{
        int ch, item;

        printf("\n Enter the size of the array : ");
        scanf("%d", &size);

        while(1)
        {
                printf("\n 1.INSERTION AT FRONT \n 2.INSERTION AT REAR \n
3.DELETION FROM FRONT \n 4.DELTION FROM REAR \n 5.DISPLAY \n 6.EXIT");
                printf("\n Enter your choice : ");
                scanf("%d", &ch);

                switch(ch)
                {
                        case 1: insertQueueFront();
                                        display();
                                        break;
                        case 2: insertQueueRear();
                                        display();
                                        break;
                        case 3: deleteQueueFront();
                                        display();
                                        break;
                        case 4: deleteQueueRear();
                                        display();
                                        break;
                        case 5: display();
                                        break;
                        case 6: exit(0);
                                        break;
                        default: printf("\n INVALID CHOICE !!!");
                }
        }
}
```

linkedList.c

```
#include<stdio.h>
#include<stdlib.h>

struct node{

        int data;
        struct node *addr;

} *head = NULL;




void traverse()
{
        struct node *ptr;

        if (head  == NULL)
        {
                printf("\n\n List is empty !!!!!!!!\n");
        }
        else
        {
                ptr = head;
                printf(" \n the list elements are :\n");
                while(ptr != NULL)
                {
                        printf(" -> %d ", ptr -> data );
                        ptr = ptr -> addr;
                }
        }
}



void insertBeg(int value)
{
        struct node *n;
        n = (struct node *) malloc (sizeof(struct node));
        n -> data = value;

        if (head == NULL)
        {
                head = n;
        }
        else
        {
                n -> addr = head;
                head = n;
        }
        printf("\n element %d successfully postioned at beginning\n", value);

        traverse();
}



void insertRandom(int value, int pos)
{
        int i;
        struct node *n, *ptr, *temp;
        n= (struct node *) malloc (sizeof(struct node));
        n -> data = value;

        ptr = head;
        for(i=1; i<pos; ++i)
        {
                temp = ptr;
```

```
                ptr = ptr -> addr;
                if (ptr == NULL)
                {
                        printf("\n no such position found!!!\n");
                        break;
                }

        }
        temp -> addr = n;
        n -> addr = ptr;
        // n -> addr = ptr -> addr;
        // ptr -> addr = n;

        traverse();
}


void insertEnd(int value)
{
        struct node *n, *ptr;
        n= (struct node *) malloc (sizeof(struct node));
        n -> data = value;


        if(head == NULL)
        {
                head = n;
        }
        else
        {
                ptr = head;
                while(ptr -> addr != NULL)
                {
                        ptr = ptr -> addr;
                }
                ptr -> addr = n;
                n -> addr = NULL;
        }

        traverse();
}




void deleteBeg()
{
        struct node *ptr;

        if (head == NULL)
        {
                printf("\n\n List is empty !!!!!!!!\n");
        }
        else
        {
                ptr = head;
                head = ptr -> addr;
                printf("\n\n element %d has been deleted from the
beginning\n", ptr -> data);
```



```
        traverse();
}



void deleteRandom(int pos)
{
```

```
        int i;
        struct node *ptr, *temp;

        if (head == NULL)
        {
                printf("\n\n List is empty !!!!!!!!\n");
        }
        else
        {
                ptr = head;
                for(i=1; i<pos; ++i)
                {
                        temp = ptr;
                        ptr = ptr -> addr;
                        if (ptr == NULL)
                        {
                                printf("\n no such position found!!!\n");
                                break;
                        }
                }
                printf("\n\n element %d has been deleted from the position %d
\n", ptr-> data, pos);
                temp -> addr = ptr -> addr;
                // temp = ptr -> addr;
                // ptr -> addr = temp -> addr;
        }

        traverse();
}



void deleteEnd()
{
        struct node *ptr, *temp;
        ptr = head;
        if (ptr -> addr == NULL)
        {
                head = NULL;
                printf("\n\n List is empty !!!!!!!!\n");
        }
        else
        {
                ptr = head;

                while(ptr -> addr != NULL)
                {
                        temp = ptr;
                        ptr = ptr -> addr;
                }
                temp -> addr = NULL;
                traverse();
        }
}



void main()
{
```



```
        // struct node* n = createNode(0);

        while(1)
        {

                printf("\n\n****MENU****\n\n");
```

```
                printf(" 1.DISPLAY\n\n 2.INSERT AT BEGINNING\n 3.INSERT AT
END\n 4.INSERT AT RANDOM\n\n 5.DELETE FROM BEGINNING\n 6.DELETE FROM END\n
7.DELETE FROM RANDOM\n\n 8.EXIT\n");
                printf("\n Enter your choice : ");
                scanf("%d", &menuChoice);

                switch(menuChoice)
                {

                        case 1:
                                        traverse();
                                        break;

                        case 2:
                                        printf("\n Enter the value to be
inserted : ");
                                        scanf("%d", &value);
                                        insertBeg(value);
                                        break;

                        case 3:
                                        printf("\n Enter the value to be
inserted : ");
                                        scanf("%d", &value);
                                        insertEnd(value);
                                        break;

                        case 4:
                                        printf("\n Enter the value to be
inserted : ");
                                        scanf("%d", &value);
                                        printf("\n Enter the position : ");
                                        scanf("%d", &pos);
                                        insertRandom(value, pos);
                                        break;

                        case 5:
                                        deleteBeg();
                                        break;

                        case 6:
                                        deleteEnd();
                                        break;

                        case 7:
                                        printf("\n Enter the position to be
deleted : ");
                                        scanf("%d", &pos);
                                        deleteRandom(pos);
                                        break;

                        case 8:
                                exit(0);

                        default :
                                        printf(" Enter a valid choice\n");

                }
        }
}
```


polynomialAddition.c

```
#include<stdio.h>
#include<stdlib.h>
```

```
struct node{
        int coeff;
        int exp;
        struct node *addr;
}*head = NULL, *h1 = NULL, *h2 = NULL, *h=NULL;


void traversal(struct node *h)
{
        struct node *ptr;
        ptr = h;

        printf("\n");
        while(ptr != NULL)
        {
                printf("   %d^%d   +", ptr -> coeff, ptr -> exp);
                ptr = ptr -> addr;
        }
        printf("\n\n");
}


struct node * createNode(int c, int e)
{
        struct node *nd;
        nd = (struct node *) malloc (sizeof(struct node));

        nd -> coeff = c;
        nd -> exp = e;
        nd -> addr = NULL;

        return nd;
}


struct node *createPoly(int s)
{
        int i,c,e;
        struct node *nd, *ptr;
        h=NULL;

        for (i = 0; i < s; ++i)
        {
                printf("\n Entre coefficient : ");
                scanf("%d", &c);
                printf(" Enter exponent : ");
                scanf("%d", &e);

                nd = createNode(c,e);

                if (h == NULL)
                {
                        h = nd;
                        ptr = nd;
                }
                else
                {
                        ptr -> addr = nd;
                        ptr = nd;
                }
```



```
        traversal(h);
        return h;
}



void polynomialAddition(struct node *p1, struct node *p2)
{
```

```
        struct node *add, *ptr;
        add = (struct node *) malloc (sizeof(struct node));

        while(p1 != NULL && p2 != NULL)
        {
                if (p1 -> exp > p2 -> exp)
                {
                        add = createNode(p1 -> coeff, p1 -> exp);
                        p1 = p1 -> addr;
                }
                else if (p2 -> exp > p1 -> exp)
                {
                        add = createNode(p2 -> coeff, p2 -> exp);
                        p2 = p2 -> addr;
                }
                else
                {
                        add = createNode(p1 -> coeff + p2 -> coeff, p1 -> exp);
                        p1 = p1 -> addr;
                        p2 = p2 -> addr;
                }

                if (head == NULL)
                {
                        head = add;
                        ptr = add;
                }
                else
                {
                        ptr -> addr = add;
                        ptr = add;
                }
        }

        while (p1 != NULL)
        {
                add = createNode(p1 -> coeff, p1 -> exp);
                p1 = p1 -> addr;

                ptr -> addr = add;
                ptr = add;
        }
        while (p2 != NULL)
        {
                add = createNode(p2 -> coeff, p2 -> exp);
                p2 = p2 -> addr;

                ptr -> addr = add;
                ptr = add;
        }
        traversal(head);
}

void main()
{
        int s1,s2;

        printf("\n Enter the size of first polynomial : ");
        scanf("%d", &s1);
        h1 = createPoly(s1);
```



```
        printf("\n Enter the size of second polynomial : ");
        scanf("%d", &s2);
        h2 = createPoly(s2);

        polynomialAddition(h1 ,h2);
}
```

postfix.c

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>


int top=-1, stack[15];


void push(char x)
{
    stack[++top] = x;
}


char pop()
{
        return stack[top--];
}


int precedence(char c)
{
        if(c == '(')
                return 0;
    else if(c == '+' || c == '-')
        return 1;
    else if(c == '*' || c == '/')
        return 2;
    else if(c == '^')
        return 3;
}


void infixToPostfix(char infix[],char postfix[])
{
    char item, x;
    int i=0, j=0;

    while(infix[i] != '\0')
    {
       item = infix[i];

        if(isalnum(item))
        {
                    postfix[j++] = item;
        }

        else if(item == '(')
        {
               push(item);
        }

        else if(item==')')
        {
                x=pop();
                while(x != '(')
                {
                        postfix[j++] = x;
                }
        }

        else
        {
               while(precedence(stack[top]) >= precedence(item))
```

```
            {
              postfix[j++] = pop();
            }
            push(item);
        }
        i++;
    }

    while(top != -1)
    {
                postfix[j++] = pop();
    }
    postfix[j] = '\0';
}


void postfixEvaluation(char postfix[])
{
    char ch;
    int a,b,i;

    for(i=0; postfix[i] != '\0'; i++)
    {
        ch = postfix[i];

        if(isdigit(ch))
        {
            push(ch-'0');
        }

        else
        {
            b = pop();
            a = pop();

            switch(ch)
            {
              case '+': push(a+b);
                break;
              case '-': push(a-b);
                break;
              case '*': push(a*b);
                break;
              case '/': push(a/b);
                break;
            }
        }
    }

    printf("\n The postfix evaluation is   : %d", stack[top]);
}


void main()
{
    char infix[15],postfix[15];

    printf("\n Enter the infix expression  : ");
    scanf("%s", infix);

    printf(" The postfix expression is   : %s",postfix);

    postfixEvaluation(postfix);

    printf("\n");
}
```